تعلم jQuery

عبد اللطيف ايمش

تعلم jQuery

ترجمة

عبد اللطيف ايمش

تقديم



أصبح ظاهرًا للعيان كيف أصبحت مكتبة jQuery من أشهر مكتبات JavaScript وأكثرها استخدامًا، إذ تستعملها كبرى الشركات كشركة Google والبرمجيات الشهيرة كبرمجية WordPress، وأظهرت الإحصائيات أنها مستعملة في أكثر من نصف مواقع الويب.

جاء هذا الكتاب لشرح المفاهيم الأساسية في jQuery والتي ستُشكِّل أساسًا للمعلومات التي ستتعلمها في المستقبل، وحاولتُ أن أتوخى فيه الدقة مع الحفاظ على بساطة الشرح ووضوحه.

هذا الكتاب مترجمٌ عن كتاب «jQuery Enlightenment» لصاحبه Cody Lindley، والذي نَشَرتهُ Syncfusion لاحقًا باسم «jQuery Succinctly». نُشِرتَ هذه النسخة المترجمة من الكتاب بعد أخذ إذن المؤلف.

وفي النهاية، أحمد الله على توفيقه لي بإتمام العمل على الكتاب، وأرجو أن يكون إضافةً مفيدةً للمكتبة العربية، والله ولي التوفيق.

عبد اللطيف محمد أديب ايمش

حلب، سورية 2017/5/3

هذا الكتاب برعاية



وادي التقنية موقعٌ تقنيٌ عربيٌ يُعنى بتتبع أخبار البرمجيات الحرة والمواد التعليمية المتعلقة بها، يكتب فيه عدد من المتطوعين المهتمين بالبرمجيات الحرة والتقنية بشكل عام؛ يهتم وادي التقنية بمواضيع مثل أنظمة التشغيل الحاسوبية والهاتفية، ولغات البرمجة، والمكتبات البرمجية، وتقنيات الويب، وأخبار شركات البرمجة الكبرى، والمصادر المفتوحة، والعتاد وأجهزة الحاسوب .

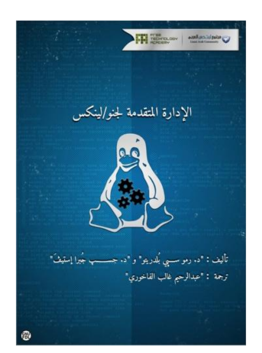
دعم وادي التقنية كتابة العديد من الكتب التقنية في مجال البرمجيات الحرة ومفتوحة المصدر، وتوفيرها مجانًا للمستخدم التقني العربي، ومن أهم الكتب التي دعمها وادي التقنية:













جدول المحتويات

الفصل الثاني: تحديد العناصر في jQuery...................61

الفصل الخامس: نماذج HTML............................117

الفصل السادس: الأحداث في jQuery.....................138

الفصل السابع: jQuery ومتصفح الويب....................167

الفصل الثامن: الإضافات في jQuery......................171

الفصل التاسع: تحسين أداء شيفرات jQuery.............189

الفصل العاشر: المؤثرات في jQuery......................203

الفصل الحادي عشر: تقنية Ajax..........................217

الفصل الثاني عشر: مواضيع متفرقة......................224

تمهيد



كتبتُ هذا الكتاب لشرح المفاهيم الأساسية لمكتبة jQuery شرحًا بسيطًا مختصرًا لكي تصبح مُبرمِجًا ذا مستوى متوسط أو متقدم، والغرضُ منه هو تعريفك على الممارسات الشائعة بين صفوف مطوري jQuery؛ ويحتوي كلُّ فصلٍ فيه على المعلومات اللازمة لكي تصبح مطوِّر jQuery محترف.

هذا الكتاب ملائمٌ لثلاثة أنواعٍ من القراء. يضمُّ أول نوعٍ مَن قرؤوا كتبًا تعريفيةً بمكتبة jQuery ويتطلعون إلى الخطوة التالية. والنوع الثاني من القراء هم مطورو JavaScript ممن لديهم خبرة وتجربة مع إحدى المكتبات ويريدون الآن تعلم jQuery بسرعة. ولا تُفاجأ إذا أخبرتُك أنَّ النوع الثالث هو «أنا»! أنشأتُ هذا الكتاب ليصبح مرجعًا شخصيًا لمفاهيم jQuery، وأرجو أن يتوافر هذا النوع من الكتب لكل مكتبةٍ من مكتبات JavaScript.

من المهم أن تتعرَّف -قبل أن نبدأ- على بعض الأمور التنظيمية المستعملة في الكتاب، أرجو ألّا تتخطى هذا القسم لأنه يحتوي على معلوماتٍ مفيدةٍ ستساعدك في الاستفادة من الكتاب.

1. اصطلاحات jQuery

المصطلح «دالة jQuery» (jQuery function) أقصد به «الدالة البانية jQuery» (أي `jQuery()` أو الشكل البديل `$()`)، وهي تُستخدَم لإنشاء نسخة من كائن `jQuery`.

المصطلح «مجموعة التغليف» (wrapper set) يُشير إلى عناصر DOM المغلَّفة (wrapped) ضمن دوال jQuery. وسيُستعمَل هذا المصطلح تحديدًا للإشارة إلى العناصر التي حُدِّدَت باستخدام دالة jQuery. ربما تسمع البعض يدعو هذا الاصطلاح «بمجموعة jQuery» (jQuery collection)، لكنني سأستخدم في هذا الكتاب المصطلح «مجموعة التغليف».

2. طريقة تنظيم هذا الكتاب

هذا الكتاب مُقسَّمٌ إلى فصولٍ مرتبةٍ بترتيب أقسام توثيق jQuery نفسه تقريبًا. يحتوي كل فصلٍ على مفاهيمٍ عن jQuery معزولةٍ عن بعضها وهي ترتبط بعنوان الفصل. إذا لم تقضِ وقتًا طويلًا في صفحات التوثيق في موقع jquery.com فأنصحك بفعل ذلك قبل قراءة هذا الكتاب.

3. شيفراتٌ أكثر وشرحٌ أقل

كتبتُ هذا الكتاب متعمِّدًا أن يتفحص القارئُ الأمثلةَ الموجودةَ فيه جيدًا. يجب أن تنظر إلى الشرح كأمرٍ ثانويٍ ملحقٍ بالشيفرة. أرى شخصيًا أنَّ الشيفرة تساوي ألف كلمة. لا تقلق إن زاد الشرح حيرتك في البداية؛ إذ عليك تفحّص الشيفرة وقراءة التعليقات مرةً أخرى، وتكرِّر هذه العملية حتى يصبح المفهوم الذي أحاول شرحه واضحًا. أرجو أن تصل إلى مرحلةٍ من الخبرة كيلا تحتاج إلا إلى شيفرةٍ موثقةٍ توثيقًا جيدًا لكي تستوعب أحد المفاهيم البرمجية.

4. لماذا أستخدم ()alert في الأمثلة؟!

أرجو أن تقتنع أنني أكره الدالة `alert()` أكثر منك، لكن صدِّق أو لا تصدق: هذه الدالة عملية وتعمل عملها بشكلٍ ممتاز في جميع المتصفحات؛ صحيحٌ أنَّه ليس من الضروري استخدامها، إلا أنني لم أشأ إضافة بعض التعقيدات الناجمة عن استخدام `console` لكي أحافظ على وضوح الشيفرات. فهدفي هو تقليل أيّة شيفرات إضافية لا تخدم غرض شرح المفهوم الذي أريد إيصاله إليك.

5. التنسيق والألوان

سأستخدمُ الخط العريض في شيفرات JavaScript (كما في المثال الآتي) للإشارة إلى الشيفرات والأسطر البرمجية التي تتعلق مباشرةً بالمفهوم الذي أشرحه، وسأستعمل اللون الفضي الفاتح للإشارة إلى التعليقات:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <!-- HTML تعليق -->
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      // JavaScript تعليق
      var focusOnThisCode = true;
    </script>
  </body>
</html>
```

بالإضافة إلى تنسيق الشيفرات، سأضيف في متن النص بعض شيفرات JavaScript، وتلك الشيفرات ستُنسَّق بخطٍ ذي عرضٍ ثابت بلونٍ فضيٍ غامق لكي تميّز بينها وبين النص العادي، مثال:

«لا ننتظر عادةً حدوثَ الحدث `window.onload`. وهذا هو الغرض من استخدام حدث خاص مثل `ready()` والذي سيؤدي إلى تنفيذ الشيفرات قبل انتهاء تحميل الصفحة [...]»

6. افهم الدالة text() في jQuery فهمًا تامًا قبل قراءة هذا الكتاب

سنستخدم الدالة `text()` التابعة لمكتبة jQuery كثيرًا في الأمثلة في هذا الكتاب. وعليك أن تنتبه إلى أنَّ الدالة `text()` -عندما تُستخدَم على مجموعةِ تغليفٍ (wrapper set) تحتوي على أكثر من عنصرٍ- ستؤدي إلى دمج عناصر مجموعة التغليف وإعادة سلسلة نصية لجميع النصوص الموجودة في كل العناصر المشكِّلة لها. ربما يشوشك هذا إذا كنتَ تتوقع أن تُعيدَ الدالةُ النصَ الموجودَ في أوَّل عنصرٍ من مجموعة التغليف. هذا مثالٌ عن كيفية دمج الدالة `text()` للسلاسل النصية الموجودة في عناصر مجموعة التغليف (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <span>I </span><span>love</span>
    <span>jQuery</span><span>!</span>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      // الناتج: "I love jQuery!"
      alert(jQuery('span').text());
    </script>
  </body>
</html>
```

7. ما هو JS Bin ولماذا أستخدمه؟

JS Bin هو تطبيق ويب صُمِّمَ خصيصًا لتسهيل تجربة الشيفرات على مطوري JavaScript و CSS.

ولمّا كان هذا الكتاب معتمدًا اعتمادًا أساسيًا على الأمثلة لشرح مفاهيم jQuery، فارتأيتُ ضرورة توفير أمثلة هذا الكتاب للتجربة المباشرة على متصفحات الويب؛ ويمكنك الوصول إلى التجربة الحية للأمثلة بالضغط على الرابط الموجود قبلها «(مثال حي)».

إتاحة الأمثلة لك مباشرةً تساعدك في تعديل والتجربة على الشيفرات من أي متصفح ويب، أنا لا أشجعك على تعديل الشيفرات فحسب، وإنما اعتمدتُ على ذلك أثناء عملي على هذا الكتاب.

الفصل الأول:

المفاهيم الأساسية في jQuery



1. المفاهيم الأساسية وراء jQuery

صحيحٌ أنَّ هنالك بعض الاختلافات في الواجهة البرمجية (API) لمكتبة jQuery (مثلًا بعض الدوال كالدالة `$.ajax`)، لكن المفهوم الرئيسي في jQuery هو «اعثر على شيءٍ ما، وافعل شيئًا ما به». تحديدًا: انتقِ عنصرًا أو عناصرَ من شجرة DOM من مستند HTML وافعل بها شيئًا ما باستخدام الدوال التي توفرها jQuery.

انظر مليًا إلى الشيفرة الآتية لتبسيط هذا المفهوم (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
      <!-- ستُغيّر jQuery هذا العنصر -->
      <a href=""></a>
      <!-- إلى هذا العنصر
        <a href="http://www.jquery.com">jQuery</a>
       -->
      <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
      <script>
        jQuery('a').text('jQuery').attr('href',
          'http://www.jquery.com');
      </script>
```

```
  </body>
</html>
```

لاحظ أننا استخدمنا jQuery في مستند HTML لتحديد العنصر <a> في شجرة DOM. وبعد تحديد شيءٍ ما سنستطيع العمل على العناصر المُحدَّدة باستخدام دوال jQuery مثل `text()` و `attr()`. الاستيعاب التام لمفهوم «اعثر على شيءٍ ما، وافعل شيئًا ما به» أساسيٌّ في تقدمك كمبرمج jQuery .

2. المفهوم وراء المفهوم الذي يقف وراء jQuery

صحيحٌ أنَّ العثور على شيءٍ ما وفعل شيءٍ به هو المفهوم الأساسي في jQuery، إلا أنني أريد توسعة هذا المفهوم لأضيف عليه «إنشاء شيءٍ ما» أيضًا. وبالتالي يمكن توسعة المفهوم وراء jQuery لكي يتضمَّن إنشاء شيءٍ جديد ثم تحديده ثم فعل شيءٍ ما به. أسمي هذا «المفهوم وراء المفهوم الذي يقف وراء jQuery».

ما أحاول فعله هو توضيح أنَّك لستَ مقيدًا بتحديد شيءٍ ما موجودٍ في شجرة DOM، فمن المهم أن تفهم أنَّ jQuery يمكن أن تُستعمَل لإنشاء عناصر DOM جديدة وفعل شيءٍ ما بها بعد ذلك.

سُننشِئ في المثال الآتي عنصرًا جديدًا هو <a> والذي لم يكن موجودًا في DOM. وسيُحدَّد بعد إنشائه وستُجرى عليه عملية (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      jQuery('<a>jQuery</a>').attr('href',
        'http://www.jquery.com').appendTo('body');
    </script>
  </body>
</html>
```

الفكرة الأساسية التي عليك فهمها هنا هي أننا أنشأنا العنصر <a> أثناء التنفيذ وتعاملنا معه كما لو أنَّه موجودٌ مسبقًا في شجرة DOM.

3. كيفية التحقق من إصدار jQuery

هنالك حالات قد لا تتمكن فيها من استخدام آخر إصدار من jQuery؛ ومن المفيد في تلك الحالات معرفة ما هو إصدار jQuery الذي تتعامل معه. من حسن الحظ أنَّ ذلك موثقٌ في الملف نفسه وفي رابط URL الذي تستخدمه لتضمين مصدر jQuery، لكن إن لم تستطع معرفة الإصدار مما سبق فيمكنك استخراج تلك المعلومات من مكتبة jQuery نفسها. سأعرض عليك أدناه حليّن للتحقق من الإصدار الذي تستعمله (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      alert(jQuery.fn.jquery);
      alert(jQuery().jquery);
    </script>
  </body>
</html>
```

4. تتطلب jQuery أن يكون مستند HTML بنمط المعايير أو نمط المعايير التقريبي

هنالك مشاكل معروفة مع دوال jQuery في أنَّها لا تعمل عملًا صحيحًا عندما يُحمِّل المتصفحُ صفحةَ HTML بنمط التجاوزات (quirks mode). تأكَّد أنَّك تستخدم jQuery في متصفحٍ يُفسِّر HTML بنمط المعايير (standards mode) أو نمط المعايير التقريبي (almost standards mode) باستخدام نوع doctype الصحيح.

للتأكد من أداء دوال jQuery لوظيفتها أداءً سليمًا، فسنستعمل نوع المستند doctype الخاص بإصدار HTML5:

```
<!DOCTYPE html>
```

5. ضمِّن جميع ملفات CSS قبل تضمين jQuery

بدءًا من jQuery 1.3، لن تَضمَن لك المكتبة أنَّ جميع ملفات CSS ستُحمَّل قبل أن تُطلِق الحدث الخاص `ready()`. وبسبب هذا التغيير في jQuery 1.3 فعليك دومًا إضافة جميع ملفات CSS قبل شيفرات jQuery. وهذا سيَضمَن تفسير المتصفح لشيفرات CSS قبل الانتقال إلى شيفرات JavaScript الموجودة لاحقًا في مستند HTML. لكن ربما تكون الصور المُشار إليها عبر CSS قد نُزِّلتَ أو لم تُنزَّل بعد عندما يُفسِّر المتصفح شيفرة JavaScript.

6. استخدام نسخة مُستضافة من jQuery

يَعمَد أغلبية الأشخاص عند تضمين jQuery في صفحات الويب إلى تنزيل الشيفرة المصدرية وربطها إلى مستند HTML عبر وضع الملف المصدري للمكتبة في موقعهم (أو نطاقهم) الشخصي. لكن هنالك خيارات أخرى التي تسمح لك باستخدام jQuery من مصدرٍ آخر.

تستضيف Google عدّة نسخ من الشيفرة المصدرية لمكتبة jQuery بغرض إتاحتها لمن يشاء استخدامها، وهذا رائع! سأستخدمُ في المثال الآتي العنصر `<script>` لتضمين نسخة مُصغّرة (minified) من jQuery والمستضافة في Google (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      alert('It is ' + ('jQuery' in window) + ' jQuery is
loaded');
    </script>
  </body>
</html>
```

تستضيف Google عدّة إصدارات من الشيفرة المصدرية لمكتبة jQuery، وهنالك نسختان لكل إصدار: واحدة مُصغّرة وأخرى غير مصغّرة. أنصحك باستخدام النسخة غير المصغّرة أثناء التطوير، لأن تنقيح الأخطاء فيها أسهل فيما إذا كنتَ تتعامل مع نسخةٍ مصغّرةٍ من الشيفرة.

ميزات استخدام النسخة المُستضافة من Google هي السرعة والوثوقية والاحتمال الكبير أن تكون المكتبة مُخزَّنةً مسبقًا على جهاز العميل.

7. تنفيذ الشيفرة عندما تجهز شجرة DOM لكن قبل الحدث window.onload

يمكن أن يُستخدَم الحَدَث (event) المضمَّن في JavaScript المسمى `window.onload` لتنفيذ شيفرات برمجية أثناء اكتمال تحميل صفحة في المتصفح. لكنَّ هذا الحدث يعني أنَّ كلَّ شيءٍ قد تم تحميله في الصفحة، ولن يُطلَق هذا الحدث إلى أن تنزَّل جميع الوسائط (مثل الصور وملفات الفيديو ...إلخ.) وسيعني انتظار حدوث هذا الحدث تضييعَ الكثير من الوقت.

فالحل الأفضل هو بدء تنفيذ الشيفرات عندما تنتهي عملية تهيئة شجرة DOM مباشرةً وتتاح إمكانية تعديلها.

توفِّر jQuery حدثًا خاصًا سيُنفَّذ بعد أن تصبح شجرة DOM صالحةً لإجراء العمليات عليها لكن قبل إطلاق الحدث `window.onload`. يُسمى هذا الحدث الخاص بمكتبة jQuery بالاسم `ready()`. وهنالك وسيطٌ وحيدٌ يمُرَّر إلى هذه الدالة ألا وهو مرجعيةٌ إلى دالةِ jQuery.

سأعطيك في الشيفرة الآتية ثلاثة أمثلة عن استخدام هذا الحدث الخاص (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>

      // الشكل القياسي
      jQuery(document).ready(function()
      {
        alert('DOM is ready!');
      });

      // الشكل المختصر، لكنه يؤدي نفس الغرض
      jQuery(function()
```

```
      {
        alert('No really, the DOM is ready!');
      });

      // الشكل المختصر مع الاستخدام الآمن للرمز $

      // أبقِ في بالك أنّه مرجعية إلى
      // دالة jQuery المُمرِّرة إلى الدالة المجهولة
      jQuery(function($)
      {
        alert('Seriously its ready!');
       /*
        استخدم $() دون أن تخاف من التضاربات مع مكتباتٍ أخرى
        */
      });
    </script>
  </head>
  <body>
  </body>
</html>
```

ملاحظات	- أبقِ ببالك أنَّك تستطيع إضافة أيِّ عددٍ تشاء من الأحداث `ready()` إلى المستند. لستَ مقيدًا بوضع حدث واحد فقط، وستُنفَّذ بترتيب إضافتها نفسه. - احرص على تضمين جميع صفحات الأنماط (CSS) قبل شيفرة jQuery، وبهذا ستَضمَن صحة عرض جميع العناصر في DOM قبل بدء تنفيذ شيفرات jQuery.

8. تنفيذ شيفرات jQuery عندما ينتهي المتصفح من تحميل كامل المستند

لا ننتظر عادةً حدوثَ الحدث `window.onload`. وهذا هو الغرض من استخدام حدث خاص مثل `ready()` والذي سيؤدي إلى تنفيذ الشيفرات قبل انتهاء تحميل الصفحة، لكن بعد أن تجهز شجرة DOM لإجراء التعديلات عليها.

لكن سنحتاج إلى الانتظار في بعض الأحيان. وصحيحٌ أنَّ الحدث الخاص `ready()` رائعٌ ويفيدنا بتنفيذ الشيفرات بمجرد أن تجهز شجرة DOM، لكننا نستطيع استخدام jQuery لتنفيذ شيفرة بعد أن تُحمَّل كامل الصفحة (بما في ذلك جميع الوسائط المُضمنة فيها) تمامًا.

يمكن فعل ذلك عبر الحدث `load` الخاص بالكائن `window`. توفِّر jQuery الدالة `on()` التي يُمكن أن تُستعمَل لاستدعاءِ دالةٍ عند وقوع حدث معيّن. سأريك مثالًا عن استخدام الدالة `on()`:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      // تمرير الكائن window إلى الدالة jQuery وربط تنفيذ
      // الدالة باكتمال تحميل المستند عند وقوع الحدث load
      jQuery(window).on('load', function()
      {
        alert('The page and all its assets are loaded!');
      });
    </script>
  </head>
  <body>
  </body>
</html>
```

ملاحظات

- يمكن أن يُستخدَم الحدث load على عناصر الصور أو إطارات iframe.

```
jQuery('img, iframe').on('load', function()
{alert('loaded');});
```

- لا تنسَ وجود الحدث unload (أو beforeunload)، والذي يمكن أن يُستعمَل لتنفيذ دوال عندما يُطلَق عند مغادرة المستخدم للصفحة.

يجدر بالذكر أننا كنّا نستعمل الدالتين ()load و ()unload من قبل، لكنهما أمستا مهملتين بدءًا من الإصدار 1.8 من jQuery، وحُذفتا بدءًا من الإصدار 3.0، ونُصِحَ باستخدام الدالة ()on بدلًا منهما، وسنشرح ذلك باستفاضة في الفصل السادس «الأحداث في jQuery».

9. تنفيذ شيفرة jQuery عندما تُفسَّر شجرة DOM، لكن دون استخدام ()ready

لا تحتاج دومًا إلى استخدام الحدث الخاص `ready()`؛ فإذا كانت شيفرة JavaScript التي كتبتها لا تؤثر على شجرة DOM فيمكنك تضمينها في أيّ مكانٍ في مستند HTML. وهذا يعني أنَّك تستطيع تفادي الحدث `ready()` تمامًا إذا لم تكن شيفرة JavaScript معتمدةً على جهازية شجرة DOM.

أغلبية شيفرات JavaScript في هذه الآونة -وخصوصًا شيفرات jQuery- تتضمن تعديلاتٍ على DOM. وبالتالي يجب أن تكون شجرة DOM مفسَّرةً من المتصفح تفسيرًا كاملًا لكي نتمكن من التعامل معها. وهذا هو السبب وراء استعمال المطورين للحدث `window.onload` في السنوات الماضية.

لتفادي استخدام الحدث `ready()` مع الشيفرات التي تتعامل مع DOM، يمكنك ببساطة وضع الشيفرة في مستند HTML قبل وسم الإغلاق للعنصر `</body>`. وبفعلك لذلك ستضمن انتهاء تحميل شجرة DOM لأنَّ المتصفح سيُفسِّر المستند من بدايته إلى نهايته؛ فإذا وضعتَ شيفرة JavaScript بعد تعريف العناصر التي تريد التعامل معها، فلا حاجةَ إذًا للحدث `ready()`.

لن أستعمل في المثال الآتي الحدث `ready()` لأنني وضعتُ الشيفرة قبل نهاية جسم المستند. وهذه هي التقنية التي سأستعملها في هذا الكتاب وفي أغلبية المواقع التي أُنشِئها (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p>Hi, I'm the DOM! Script away!</p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      alert(jQuery('p').text());
    </script>
  </body>
</html>
```

إذا وضعتُ العنصر <script> قبل العنصر <p> فستُنفَّذ الشيفرة قبل تحميل المتصفح للعنصر <p>، وهذا سيجعل jQuery تفترض عدم احتواء المستند على أيّة عناصر <p>؛ لكن إذا استعملتُ الحدث الخاص ready() فعندئذٍ لن تُنفِّذ jQuery الشيفرةَ حتى تجهز شجرة DOM تمامًا. لكن لماذا نستعمل الحدث ready() إن أمكننا التحكم في مكان عنصر <script> في المستند؟ فوضعنا لشيفرات jQuery في أسفل الصفحة سيؤدي إلى تجنب استخدامنا للحدث ready()؛ وإذا ابتغينا الدقة فإنَّ وضع جميع شيفرات JavaScript في أسفل الصفحة سيؤدي إلى تحسين أدائها.

10. استخدام $ دون القلق من حدوث تضاربات

تستخدم jQuery المحرف $ للوصول إلى مجال أسماء jQuery (jQuery namespace)، لكننا

لسنا وحدنا الذين نحب المحرف $، لأنَّ بقية المكتبات تستخدم المحرف $ أيضًا! لكن ليس عليك التخلي تمامًا عن استخدامه...

يمكنك الاستمرار في استخدام $ عبر استعمال دالة مجهولة تُنفَّذ مباشرةً وتمرير الكائن jQuery إليها. وبعدئذٍ ستتمكن من استخدام المحرف $ داخل تلك الدالة للإشارة إلى jQuery، وبالتالي سيُنشَأ مجالٌ فريدٌ (unique scope)، ويُعرَف أيضًا بالتعبير المغلق أي (closure). أترجم ما سبق إلى العربية: يمكنك الحصول على مجالٍ خاصٍ يمكنك فيه استخدام $ بحرية دون القلق من التضارب مع مكتبات JavaScript الأخرى التي قد تتواجد في نفس مستند HTML.

يحتوي المثال الآتي على تطبيقٍ للمفهوم السابق (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
       // استخدام $() دون الخوف من التضارب مع مكتباتٍ أخرى
        alert('You are using jQuery ' + $().jquery);
      })(jQuery)
    </script>
```

```
    </body>
</html>
```

ملاحظة	لا تُغفِل وجود الدالة `jQuery.noConflict()` التي تعيد ربط الكائن الأصلي (التابع لمكتبةٍ أخرى) الذي كان مُشارًا إليه بالمحرف `$` إلى `$` مرةً أخرى (إذا كنتَ تستعمل أكثر من مكتبة تستخدم المحرف `$`). وهذا يسمح لك باستخدام `$` الخاص بالمكتبة الأخرى بالإضافة إلى إمكانية استخدامك لمكتبة jQuery والتي ستتمكن من الوصول إليها عبر الكائن `jQuery()` فقط بدلًا من `$()`.

11. فهم كيفية عمل السلاسل في jQuery

بعد أن تُحدِّد شيئًا باستخدام الدالة jQuery وتُنشِئ مجموعة تغليف، فستتمكن من إنشاء سلسلةٍ من دوال jQuery للتعامل مع عناصر DOM الموجودة داخل المجموعة. ما تفعله jQuery هو جعل الدوال الموجودة في السلسلة تُعيد مجموعة التغليف في كل مرة ينتهي فيها تنفيذ إحداها، وتلك المجموعة ستُستخدَم من قِبل الدالة التالية في السلسلة. لاحظ أنَّ أغلبية الدوال في jQuery يمكن وضعها في سلسلة، إلا أنَّ هنالك بعض الدوال التي «ستكسر» السلسلة ولن تستطيع إكمال السلسلة بعد استخدامها.

يجب أن تحاول دومًا إعادة استخدام مجموعة التغليف باستخدام سلسلة من الدوال. سأريك في المثال الآتي كيفية إنشاء سلسلة من الدوال `text()` و `attr()` و `addClass()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a></a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('a')
          // ستعيد: $('a')
          .text('jQuery')
          // ستعيد: $('a')
          .attr('href', 'http://www.jquery.com/')
          // ستعيد: $('a')
          .addClass('jQuery');
      })(jQuery)
    </script>
  </body>
</html>
```

12. كسر السلسلة باستخدام دوال «هادمة»

كما ذكرتُ سابقًا، لا يمكن لكل دوال jQuery الحفاظ على السلسلة. فهنالك دوال مثل `text()` التي يمكن أن تُستخدَم في وسط السلسلة عندما «تَضبِط» القيمة النصية للعنصر، لكن الدالة

`text()` ستؤدي إلى «كسر» السلسلة عندما «تَستخرِج» أو «تحصل على» القيمة النصية الموجودة داخل عنصر.

سأستخدمُ في المثال الآتي الدالة `text()` لضبط القيمة النصية للعنصر `<a>` ومن ثم للحصول عليها (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p></p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        var theText = $('p')
          .text('jQuery')

          // ستُعيد السلسلة النصية "jQuery"
          .text();

        alert(theText);
        // لا يمكن إكمال السلسلة بعد text() لأن السلسلة قد
        // انقطعت حيث أُعيدَت سلسلةٌ نصيةٌ، وليس كائن jQuery
      })(jQuery)
```

```
    </script>
  </body>
</html>
```

الحصول على النص الموجود ضمن عنصر باستخدام `text()` هو أبسط مثالٍ عن كيفية «كسر» السلسلة لأنَّ إحدى الدوال (التي هي `text()` في مثالنا) أعادت سلسلةً نصيةً تحتوي القيمة النصية للعقدة (node)، ولم تُعِد مجموعة التغليف الخاصة بكائن jQuery.

يجب ألّا يُدهشك أنَّه إذا لم تُعِد إحدى دوال jQuery مجموعةَ التغليف، فستُكسر السلسلة، ونَعتَبِر هذه الدوال أنّها دوالٌ «هادمةٌ» (destructive).

13. استخدام الدوال الهادمة في jQuery والعودة منها باستخدام end()

تُعتبَر الدوال الموجودة في jQuery والتي تُعدِّل مجموعة التغليف الأصلية في jQuery أنَّها دوالٌ هادمةٌ (destructive). السبب وراء ذلك هو عدم حفاظها على الحالة الأصلية لمجموعة التغليف؛ لكن لا تقلق لن «يُهدَم» شيءٌ أو يُحذَف. وإنما ستُربَط مجموعة التغليف السابقة بمجموعةٍ جديدة (new set).

على أيّة حال، لا يتوقف تعاملنا مع السلاسل عندما تُغيّر مجموعة التغليف الأصلية. فباستخدامنا للدالة `end()` سنتمكن من «العودة» من أيّة تعديلات «هادمة» أُجريَت على مجموعة التغليف الأصلية. انظر مليًا إلى استخدام الدالة `end()` في المثال الآتي لتفهم كيفية التعامل معها (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <style>
      .last {
        background: #900
      }
    </style>
    <ul id="list">
      <li></li>
      <li>
        <ul>
          <li></li>
          <li></li>
          <li></li>
        </ul>
      </li>
      <li></li>
      <li></li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
```

```
        $('#list') // مجموعة التغليف الأصلية
          .find('> li') // دالة هادمة
            .filter(':last') // دالة هادمة
              .addClass('last')
            .end() // نهاية .filter(':last')
            .find('ul') // دالة هادمة
              .css('background', '#ccc')
                .find('li:last') // دالة هادمة
                  .addClass('last')
                .end() // نهاية .find('li:last')
              .end() // نهاية .find('ul')
            .end() // نهاية .find('> li')
          .find('li') // العودة إلى $('#list')
            .append('I am an &lt;li&gt;');
      })(jQuery);
    </script>
  </body>
</html>
```

14. الدالةُ jQuery متعددةُ الاستخدامات

الدالـة jQuery هي دالـةٌ متعـددة الاسـتخدامات، فيمكننـا تمريـر مختلـف القيم وتنسـيقات السلاسـل النصـية إليهـا ومن ثم سـنتمكن من اسـتخدامها لتنفيـذ وظـائف معيّنـة. هـذه بعض استخدامات الدالة jQuery:

- تحديد العناصر من شجرة DOM باستخدام تعابير شبيهة بمحدِّدات CSS، بالإضافة إلى إمكانية استخدام تعابير خاصة بمكتبة jQuery؛ والقدرة على تحديد العناصر عبر مكانها في شجرة DOM: `jQuery('p > a')` أو `jQuery(':first')` أو `jQuery(document.body)`
- إنشاء عناصر HTML أثناء التنفيذ، بتمرير سلسلة نصية تحتوي على شيفرة HTML التي تحتوي على البُنى أو العناصر التي تريد إنشاءها، أو عبر دوال DOM التي تُنشِئ عناصر DOM: `jQuery('<div id="nav"></div>')` أو `jQuery(document.createElement('div'))`
- اختصار للحدث `ready()` وذلك عند تمرير دالة إلى الدالة jQuery: `jQuery(function($){ /* code */ })`

كلُّ نقطةٍ من النقاط السابقة موضَّحةٌ في المثال الآتي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      // تمرير دالة إلى jQuery
      jQuery(function($)
      {
```

```
        // HTML تمرير سلسلة نصية فيها
        $('<p></p>').appendTo('body');
        // jQuery تمرير مرجعية إلى عنصر إلى الدالة
        $(document.createElement('a'))
          .text('jQuery').appendTo('p');
        // jQuery تمرير تعبير CSS إلى الدالة
        $('a:first').attr('href', 'http://www.jquery.com');
        // jQuery تمرير مرجعية DOM إلى الدالة
        $(document.anchors[0]).attr('jQuery');
      });
    </script>
  </body>
</html>
```

من الممكن أيضًا تمرير مصفوفةٍ من العناصرِ إلى الدالة jQuery، وإن لم يكن ذلك شائعًا (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <ul>
      <li>dog</li>
      <li>cat</li>
    </ul>
    <script type="text/JavaScript"
```

```
src="http://code.jquery.com/jquery-latest.js"></script>
    <script>
      // الحصول على مصفوفة من كائنات li، وستُنهى السلسلة
      var liElements = jQuery('ul li').get();

      // الناتج: "dog"
      alert(jQuery(liElements).eq(0).text());
    </script>
  </body>
</html>
```

15. معرفة متى تُشير الكلمة المحجوزة this إلى كائنات DOM

عندما تُربَط الأحداثُ إلى عناصرَ DOM موجودةٍ ضمنَ مجموعةِ تغليفٍ، فيمكن استخدام الكلمةِ المحجوزةِ this للإشارة إلى كائن DOM الذي استدعى الحدث. المثال الآتي يحتوي شيفرة jQuery والتي ستَربِط الحدث الخاص (والموجود في مكتبة jQuery) mouseenter إلى كل عناصر <a> في الصفحة. هنالك حدثٌ مُضمَّنٌ في لغة JavaScript باسم mouseover والذي سيُطلَق عندما يدخل مؤشر الفأرة أو يخرج من عنصرٍ ابن (child element)، بينما لا يفعل ذلك الحدث الخاص mouseenter وهذا هو الفرق بينهما (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a id="link1">jQuery.com</a>
```

```
    <a id="link2">jQuery.com</a>
    <a id="link3">jQuery.com</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('a').mouseenter(function()
        {
          alert(this.id);
        });
      })(jQuery);
    </script>
  </body>
</html>
```

استخدمنا الكلمة المحجوزة `this` داخل الدالة المجهولة (anonymous function) التي مُرِّرَت إلى الدالة `mouseenter()` للإشارة إلى عنصر `<a>` الحالي. وفي كل مرة ستلمس فيها الفأرة النص «jQuery.com» فسيُظهِر المتصفح تحذيرًا (عبر الدالة `alert()`) يبيّن أيُّ العناصر مرَّت الفأرة فوقها عبر عرض قيمة الخاصية `id`.

كان من الممكن في المثال السابق تمرير `this` إلى دالة jQuery لكي يُغلّف عنصر DOM داخل jQuery، فبدلًا من:

```
// DOM الوصول إلى خاصية ID لعنصر
alert(this.id);
```

كنا نستطيع فعل هذا:

```
// jQuery تغليف عنصر DOM داخل كائن
// ID ومن ثم سنتمكن من استخدام attr() للوصول إلى قيمة
alert($(this).attr('id'));
```

ما سبق ممكنٌ لأنَّ الدالة jQuery لا تقبل تمرير تعابير لتحديد العناصر فحسب، وإنما تقبل أيضًا تمرير مرجعية (reference) إلى كائنات DOM. حيث تُشير الكلمة المحجوزة this في الشيفرة السابقة إلى عنصرٍ في شجرة DOM.

السبب وراء رغبتك باستخدام jQuery مع كائنات DOM يجب أن يكون واضحًا وبدهيًا؛ ففعل ذلك سيعطيك إمكانية استخدام دوال jQuery وتطبيقها على شكل سلسلة على تلك الكائنات إن احتجتَ إلى ذلك.

المرور على مجموعة من العناصر الموجودة ضمن مجموعة تغليف في jQuery هو أمرٌ يشابه المفهوم الذي ناقشناه سابقًا. فيمكننا المرور على كل عنصرٍ من عناصر شجرة DOM والموجود في مجموعة تغليف باستخدام الدالة `each()` في jQuery. وهذا ما يمنحنا وصولًا إلى كل عنصر DOM على حدة عبر استخدام الكلمة المحجوزة `this`.

بناءً على شيفرة المثال السابق، ستُحدِّد في هذا المثال جميع عناصر <a> في الصفحة وسنستخدم الدالة `each()` للمرور على كل عنصر <a> في مجموعة التغليف، ومن ثم سنحاول

الوصول إلى خاصية id التابعة لتلك العناصر. هذا هو المثال النهائي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a id="link1">jQuery.com</a>
    <a id="link2">jQuery.com</a>
    <a id="link3">jQuery.com</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // id حلقة تكرار التي ستُظهِر قيمة
        // لكل عنصر <a> في الصفحة
        $('a').each(function()
        {
          // تُشير إلى العنصر الحالي في حلقة التكرار this
          alert($(this).attr('id'));
        });
      })(jQuery);
    </script>
  </body>
</html>
```

إذا جرّبتَ المثال السابق في متصفحٍ، فسيُظهِر تحذيرًا (alert) فيه قيمة الخاصية `id` لكل عنصر `<a>` في الصفحة؛ ولوجود ثلاثة عناصر `<a>` في الصفحة فستمرّ حلقة التكرار ثلاث مرات عبر الدالة `each()` وستظهر ثلاث نوافذ تحذير.

16. استخراج العناصر من مجموعة التغليف لاستخدامها مباشرةً دون jQuery

إذا حدَّدتَ عناصر HTML باستخدام jQuery فهذا لا يعني أبدًا أنك ستفقد الوصول إلى عناصر DOM نفسها. يمكنك دومًا استخراج عنصر من مجموعة التغليف والتعامل معه عبر لغة JavaScript. على سبيل المثال، سأضبط -في المثال الآتي- خاصية `title` لعنصر `<a>` في صفحة HTML بتعديل قيمة الخاصية `title` الأصلية لعنصر DOM (مثال حي).

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a>jQuery.com</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // title لضبط خاصية DOM استخدام خاصيات
        $('a').get(0).title = 'jQuery.com';
```

```
        // تعديل عنصر DOM باستخدام دوال jQuery
        $('a').attr('href', 'http://www.jquery.com');
      })(jQuery);
    </script>
  </body>
</html>
```

وكما شرحنا سابقًا، توفِّر jQuery الدالة `get()` للوصول إلى عناصر DOM التي لها فهرسٌ (index) معيّنٌ في مجموعة التغليف.

لكن هنالك خيارٌ آخرٌ هنا. إذ يمكنك تفادي استخدام الدالة `get()` عبر استخدام الأقواس المربعة على كائن jQuery نفسه (والتي تُستعمَل للوصول إلى عناصر المصفوفات). ففي سياق مثالنا السابق، يمكن تعديل الشيفرة:

```
$('a').get(0).title = 'jQuery.com';
```

لتصبح كالآتي:

```
$('a')[0].title = 'jQuery.com';
```

ستتمكن في كلا الطريقتين من الوصول إلى عنصر DOM. شخصيًا أفضِّل استخدام الأقواس المربعة، لأنَّها أسرع إذ سنستخدم حينئذٍ JavaScript للحصول على عنصرٍ من عناصر المصفوفة، بدلًا من الحاجة إلى استخدام إحدى دوال jQuery.

لكن الدالة `get()` ستفيدك إن شئتَ أن تضع جميع عناصر DOM في مصفوفة. حيث سيؤدي استدعاء الدالة `get()` دون تمريرِ فهرسٍ إليها إلى إعادة جميع عناصر DOM الموجودة في مجموعة التغليف على شكل مصفوفة عادية.

لنختبر الدالة `get()` لإيضاح ما سبق، حيث سأضع جميع عناصر `<a>` في المثال الآتي ضمن مصفوفة، ثم سأستخدم المصفوفة للوصول إلى خاصية `title` لعنصر `<a>` الثالث في الصفحة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a href="http://www.jquery.com"
title="anchor1">jQuery.com</a>
    <a href="http://www.jquery.com"
title="anchor2">jQuery.com</a>
    <a href="http://www.jquery.com"
title="anchor3">jQuery.com</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // إنشاء مصفوفة من مجموعة التغليف
        var arrayOfAnchors = $('a').get();
```

```
        // إظهار خاصية title للرابط الثالث
        alert(arrayOfAnchors[2].title);
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة

استخدام `get()` سيؤدي إلى «كسر» سلسلة دوال jQuery. حيث ستأخذ هذه الدالةُ مجموعةَ التغليف وستحولها إلى مصفوفةٍ من عناصر DOM التي لم تعد مُضمَّنَةً في كائن jQuery. وبالتالي استخدام الدالة `.end()` لن يستعيد السلسلة بعد استخدام `.get()`.

17. التحقق إن كانت مجموعة التغليف فارغة

قبل أن تبدأ بالتعامل مع مجموعة التغليف، من المنطقي أولًا التحقق من أنَّك قد حدَّدَت شيئًا ما! أبسط حل هو استخدام العبارة الشرطية `if` للتأكد إن كانت مجموعة التغليف تحتوي أيّة عناصر DOM (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a>jQuery</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
```

```
y.min.js"></script>
    <script>
      // هل هنالك عناصر في المجموعة؟
      if (jQuery('a').get(0))
      {
        jQuery('a').attr('href', 'http://www.jquery.com');
      }
      // التحقق من طول المجموعة؛ يمكن استخدام .size() أيضًا
      if (jQuery('a').length)
      {
        jQuery('a').attr('title', 'jQuery');
      }
    </script>
  </body>
</html>
```

في الحقيقة إنَّ عبارات `if` السابقة ليست ضروريةً، لأنَّ jQuery ستفشل بصمت إن لم يُعثَر على أيّة عناصر، ولكن ستُستدعى كل دالةٍ مرتبطةٍ بمجموعةِ التغليفِ الفارغة. وصحيحٌ أننا نستطيع تجنب استعمال العبارة الشرطية `if`، لكن ذلك مستحسن؛ فاستدعاء الدوال على مجموعة تغليف فارغة قد يستهلك وقت معالجة إضافي، وقد تحصل على نتائج غير مرغوب فيها إن أعادت تلك الدوال قيمًا غير متوقعة ومن ثم أجريتَ عملياتٍ على تلك القيم.

18. إنشاء اسم بديل للوصول إلى الدالة jQuery

توفِّر مكتبة jQuery الدالة `noConflict()` التي لها عدّة استخدامات. أحد أبرز استخداماتها هو القدرة على استخدام اسم بديل آخر بدلًا من `$`. وقد تستفيد من هذا بثلاث طرائق: يمكنك إعادة ربط المحرف `$` بمكتبة أخرى تستعمله دونًا عن jQuery، ويساعدك ذلك في تجنب التضاربات، ويوفِّر لك إمكانية تخصيص اسمٍ بديلٍ (alias) لكائن jQuery.

لنقل أنَّك تطوِّر تطبيق ويب لشركة باسم XYZ، فربما تجد من المناسب أن تُخصِّص jQuery لكي تستبدل `XYZ('div').show()` بالتعبير `jQuery('div').show()` أو `$('div').show()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>XYZ company</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      var XYZ = jQuery.noConflict();
      // jQuery تجربة إحدى دوال
      alert(XYZ("div").text());
    </script>
  </body>
</html>
```

ملاحظة	بتمرير قيمة منطقية (boolean) تساوي `true` إلى الدالة `noConflict()` فستتمكن من فكّ ارتباط المتغير `jQuery` بالمكتبة، ولا يجدر بك فعل ذلك إلا في بعض الحالات النادرة مثل استخدام أكثر من إصدار jQuery في نفس الصفحة، لأنَّ ذلك قد يُسبِّب مشاكل مع إضافات jQuery.

19. استخدام ()each. عندما لا يكون الدوران الضمني على العناصر كافيًا

آمل أنَّ يكون واضحًا بالنسبة إليك أنَّه إذا كانت لديك صفحة HTML (كالظاهرة في المثال أدناه) وفيها ثلاثة عناصر `<div>` فارغة فإنَّ تعليمة jQuery التالية ستُحدِّد العناصر الثلاثة في الصفحة، وتمر عليها (دوارًا ضمنيًا [implicit iteration]) وستُضيف السلسلة النصية «I am a div» لكل العناصر الثلاثة (أي عناصر `<div>`) (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div></div>
    <div></div>
    <div></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
```

```
      (function($)
      {
        $('div').text('I am a div');
      })(jQuery);
    </script>
  </body>
</html>
```

اعتبرنا أنَّ الدوران ضمنيٌ لأنَّ شيفرة jQuery تفترض أنَّك تريد معالجة العناصر الثلاثة، وهذا يتطلب المرور على كل عنصر وضبط القيمة النصية لكل <div> إلى النص «I am a div». وهذا ما يُسمى بالدوران الضمني.

ما سبق مفيدٌ جدًا، وأغلبية دوال jQuery تستعمل الدوران الضمني. لكن ستُطبَّق بعض الدوال على أوّل عنصر في مجموعة التغليف فقط. على سبيل المثال، الدالة attr() التابعة لمكتبة jQuery ستصل إلى أول عنصر في مجموعة التغليف عندما تُستخدَم للحصول على قيمة إحدى الخاصيات.

ملاحظة	عند استخدام الدالة attr() لضبط خاصية، فستُطبِّق jQuery دورانًا ضمنيًا لضبط الخاصية لجميع العناصر الموجودة في مجموعة التغليف.

تحتوي مجموعة التغليف في الشيفرة الآتية على جميع عناصر <div> الموجودة في الصفحة، لكن الدالة attr() ستُعيد قيمة id لأول عنصر موجود في المجموعة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="div1">I am a div</div>
    <div id="div2">I am a div</div>
    <div id="div3">I am a div</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // ستظهر قيمة الخاصية لأوّل عنصر في مجموعة التغليف
        alert($('div').attr('id')); // الناتج: "div1"
      })(jQuery);
    </script>
  </body>
</html>
```

وللتوضيح، افترضُ أنّ ما تُريدُه هو الحصول على قيمة الخاصية id لكل عنصر في الصفحة. يمكنك أن تكتب ثلاث تعليمات jQuery للوصول إلى قيمة خاصية id لكل عنصر <div>. ستبدو الشيفرة عندئذٍ كالتالية:

```
$('#div1').attr('id');
$('#div2').attr('id');
```

```
$('#div3').attr('id');

// أو نُخزِّن الطلبية
var $divs = $('div');
$divs.eq(0).attr('id'); // البدء من العدد 0 وليس 1
$divs.eq(1).attr('id');
$divs.eq(2).attr('id');
```

ألا ترى أنَّ الشيفرة السابقة طويلة جدًا؟ أليس من الأفضل أن نمر بحلقة تكرار على عناصر مجموعة التغليف ومن ثم نستخرج قيمة الخاصية id ببساطة من كل عنصر <div>؟ يمكننا باستخدام الدالة $().each() أن نجري تكرارًا على عناصر مجموعة التغليف عندما نحتاج إلى إنشاء حلقة تكرار يدويًا (وليس ضمنيًا) للتعامل مع عدّة عناصر.

سأستخدمُ في المثال الآتي الدالة $().each() للمرور على عناصر مجموعة التغليف، ومن ثم الوصول إلى كل عنصر في المجموعة واستخراج قيمة خاصية id التابعة له (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="div1">div1</div>
    <div id="div2">div2</div>
    <div id="div3">div3</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
```

```
y.min.js"></script>
    <script>
      (function($)
      {
        // إظهار 3 تحذيرات
        $('div').each(function()
        {
          // this تمثِّل كل عنصر في مجموعة التغليف
          alert($(this).attr('id'));
          // alert(this.id) :يمكن أن نكتب أيضًا
        });
      })(jQuery);
    </script>
  </body>
</html>
```

تخيّل ما هي الإمكانيات التي تتاح لك عندما تتمكن من إجراء عملية التكرار يدويًا في أيّ وقتٍ تريد!

ملاحظة	توفِّر jQuery الدالة `$.each` ولا تخلط بينها وبين الدالة `$().each` التي تُستخدم للدوران على عناصر مجموعة تغليف تابعة للكائن jQuery. إذ يمكن أن تُستخدَم الدالة `$.each` للدوران على مصفوفة أو كائن عادي في JavaScript؛ أي أنها بديلٌ عن حلقات التكرار الموجودة في لغة JavaScript.

20. ستُعاد العناصر في مجموعة تغليف jQuery بنفس ترتيب ورودها في المستند

في إصدار 1.3.2 jQuery وما بعده، سيُعيد مُحرِّكُ التحديد النتائجَ كما هي مرتبةٌ في المستند وليس كما هو ترتيبها في المُحدِّد (selector) الذي مُرِّر إلى دالة jQuery. أي أنَّ محتويات مجموعة التغليف ستكون بنفس ترتيب ورود العناصر في المستند من الأعلى إلى الأسفل. لكن الحال لم يكن كما هو عليه قبل إصدار 1.3.2 (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <h1>h1</h1>
    <h2>h2</h2>
    <h3>h3</h3>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // مرّرنا h3 أولًا، لكن العنصر h1 موجودٌ في بداية
        // المستند لذلك سيُذكَر أولًا في مجموعة التغليف
        // الناتج: "H1"
        alert($('h3, h2, h1').get(0).nodeName);
      })(jQuery);
```

```
    </script>
  </body>
</html>
```

21. تحديد ما هو السياق المُستخدَم من دالة jQuery

السياق (context) الافتراضي المُستخدم من دالة jQuery عند تحديد عناصر DOM هو العنصر `document` (مثلًا: `$('a', document)`) وهذا يعني أنَّك إن لم توفِّر معامِلًا (parameter) ثانيًا لدالة jQuery (مثلًا: `jQuery()`) لتحديد ما هو السياق الذي ستُنفَّذ فيه طلبية DOM، فسيكون السياق الافتراضي هو العنصر `document`، والمعروف عمومًا بالعنصر `<body>`.

من الممكن معرفة السياق الذي تُجري فيه دالة jQuery طلبية DOM باستخدام الخاصية `context`. سأريك فيما يلي مثالين عن الحصول على قيمة الخاصية `context` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>
      <div>
        <div id="context">
          <a href="#">jQuery</a>
        </div>
      </div>
    </div>
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery
.min.js"></script>
    <script>
      (function($)
      {
        // "object HTMLDocument" :الناتج
        // $('a', document).context; يمكنك أيضًا استخدام
        alert($('a').context);
        // "object HTMLDivElement" :الناتج
        alert($('a', $('#context')[0]).context);
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة	بدءًا من الإصدار 1.10 من jQuery، أصبحت الخاصية `.context` مهملةً وحُذِفَت في الإصدار 3.0، لذا تجنب استخدامها في التطبيقات التي تستعمل نسخًا حديثةً من jQuery.

22. إنشاء بُنية DOM كاملة مع أحداثها في سلسلةٍ وحيدة

يمكنك عبر استخدام دوال jQuery في سلسلة أن تُنشِئ بُنى DOM كاملة بدلًا من إنشاء عنصر DOM وحيد فقط. سأُنشِئ في المثال الآتي قائمةً غيرَ مرتبةٍ من الروابط (لصفحات jQuery) ثم سأضيفها إلى شجرة DOM (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {

        jQuery('<ul></ul>')
          .append('<li><a>jQuery.com</a></li><li>
            <a>jQueryDocumentation</a></li>')
            .find('a:first')
              .attr('href', 'http://www.jquery.com')
          .end()
          .find('a:eq(1)')
            .attr('href', 'http://docs.jquery.com')
          .end()
          .find('a')
            .click(function()
            {
              return confirm('Leave this page?');
            })
          .end()
          .appendTo('body');
```

```
      })(jQuery);
    </script>
  </body>
</html>
```

المفهـوم الـذي عليـك اسـتيعابه من المثـال السـابق هـو أنَّ jQuery يمكن أن تُسـتعمَل لإنشـاء والتعامل مع بُنى DOM معقدة. ويمكنك باستخدام دوال jQuery فقط أن تُنشِئ بنى DOM التي تحتاج لها مهما بلغت من التعقيد.

الفصل الثاني:

تحديد العناصر في jQuery



1. يمكن لمُرشِّحات jQuery الخاصة أن تُحدِّد العناصر إن اُستعمِلَت بمفردها

ليس من الضـــــروري توفـــير عنصـــر عنـــد اســـتخدام المرشـــحات (filters) مثل `$('div:hidden')`، فمن الممكن ببساطة تمرير المُرشِّح بمفرده في أي مكان يمكنك وضع تعبيرٍ للتحديدِ فيه.

بعض الأمثلة:

```
// تحديد جميع العناصر المخفية
$(':hidden');
// تحديد جميع عناصر div، ثم اختيار ذوات الترتيب الفردي
$('div').filter(':even');
```

2. تمييز الفرق بين المرشِّحَين :hidden و :visible

لا يأخذ المُرشِّحان `:hidden` و `:visible` الخاصان بمكتبة jQuery خاصيةَ `visibility` في CSS بالحسبان كما قد تتوقع. فالطريقة التي تستعملها jQuery لتحديد إن كان العنصر مخفيًا (hidden) أو ظاهرًا (visible) هي معرفة إن كان يَشغلُ العنصرُ أيّة مساحةٍ في المستند. ولتوخي الدقة، يكون العنصر ظاهرًا إذا أعاد المتصفح قيمةً أكبر من الصفر لإحدى الخاصيتين `offsetWidth` أو `offsetHeight`. وبهذه الطريقة يمكن أن يملك أحد العناصر الخاصية `display` في CSS ذات القيمة `block` لكنه محتوى في عنصرٍ له الخاصية `display` وقيمتها none، وبهذا لن يكون العنصر ظاهرًا.

تُفحّص الشيفرة الآتية بعناية وتأكّد أنّك تفهم لماذا القيمة المُعادة هي `true` على الرغم من أننا ضبطنا الخاصية `display: block` للعنصر `<div>` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="parentDiv" style="display:none;">
      <div id="childDiv" style="display:block;"></div>
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج true، لأن عنصر <div> الأب مخفيٌ
        // لذا فستكون قيمة الخاصيتان offsetWidth
        // و offsetHeight لعنصر <div> الداخلي مساويةً للصفر
        alert($('#childDiv').is(':hidden'));
      })(jQuery);
    </script>
  </body>
</html>
```

3. استخدام الدالة is() لإعادة قيمة منطقية

من الضروري أحيانًا معرفة إن كانت تحتوي مجموعة من العناصر على عنصرٍ مُعيّن. يمكننا استخدام تعبير أو مُرشِّح (filter) للتحقق من احتواء المجموعة الحالية عليه وذلك بوساطة الدالة `is()`. ستُعاد القيمة `true` إذا احتوت المجموعة على عنصرٍ واحدٍ على الأقل يُطابِق التعبير أو المُرشِّح الذي استخدمناه. وإن لم تحتوي المجموعة على ذلك العنصر فستُعاد القيمة `false`. أمعن النظر في المثال الآتي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="i0">jQuery</div>
    <div id="i1">jQuery</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج true
        alert($('div').is('#i1'));

        // الناتج false
        // لأن مجموعة التغليف لا تحتوي على <div> فيه id="i2"
        alert($('div').is('#i2'));
```

```
        // false الناتج
        // لأن مجموعة التغليف لا تحتوي على <div> مخفي
        alert($('div').is(':hidden'));
      })(jQuery);
    </script>
  </body>
</html>
```

يجب أن يكون جليًا لك أنَّ الدالة `alert()` الثانية ستعيد `false` لعدم احتواء مجموعة التغليف في مثالنا على عنصر `<div>` له خاصية `id` ذات القيمة `i2`. الدالة `is()` مفيدةٌ جدًا إذا أردتَ معرفة إن احتوت مجموعة التغليف على عنصرٍ معين.

ملاحظات

- اعتبارًا من jQuery 1.3، أصبحت الدالة `is()` تدعم جميع التعبيرات. ففيما سبق كانت التعبيرات المعقدة -كتلك التي تحتوي على مُحدِّدات للتسلسل الهرمي (مثل + و ~ و >)- تُعيد القيمة `true` دائمًا.

- تُستعمَل الدالة `filter()` من قِبَل دوال jQuery الداخلية الأخرى، لذا أيّة قواعد تنطبق عليها هناك ستنطبق هنا أيضًا.

- يستعمل بعض المطورون الدالةَ `is('.class')` لتحديد إن كان يملك عنصرٌ ما فئةَ CSS مُعيّنة، لكن لا تنسَ أنَّ jQuery تملك دالةً لفعل ذلك اسمها `hasClass('class')` التي يمكن أن تُستعمَل على العناصر التي تحتوي على أكثر من فئة. لكن في الحقيقة، الدالة `hasClass()` تستعمل الدالة `is()` داخليًا.

4. يمكنك أن تُمرِّر أكثر من مُحدِّد إلى jQuery

يمكنـك توفـير عـدّة تعبـيرات كـأول معامـل إلى الدالـة jQuery مفصـولٌ بينهـا بفاصـلة: `$('expression, expression, expression')`، بعبارةٍ أخرى، لستَ مقيدًا بتحديد العناصر عبر تعبيرٍ وحيد. سأمرِّر على سبيل المثال في الشيفرة الآتية ثلاثة تعبيرات إلى الدالة jQuery مفصولٌ بينها بفاصلة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>jQuery </div>
    <p>is the </p>
    <ul>
      <li>best!</li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // jQuery is the best! الناتج
        alert($('div, p, ul li').text());
```

```
        // jQuery is the best! الناتج ،طريقة غير فعالة
        alert($('div').text() + $('p').text() +
          $('ul li').text());
      })(jQuery);
    </script>
  </body>
</html>
```

كلُّ تعبيرٍ من التعابير السابقة سيُحدِّد عناصرَ DOM معيّنة والتي ستُضاف جميعًا إلى مجموعة التغليف. يمكنك بعد ذلك التعامل مع تلك العناصر باستخدام دوال jQurey. أبقِ ببالك أنَّ جميع العناصر المُحدَّدة ستتواجد في نفس مجموعة التغليف. يمكن إظهار نفس الناتج لكن بطريقة غير فعالة وليس ذات كفاءة عبر استدعاء الدالة jQuery ثلاث مرات: مرّةً لكل تعبير.

5. معرفة أنَّ عنصرًا ما قد حُدِّدَ

من الممكن معرفة إن حَدَّد التعبيرُ أحدَ العناصر بالتحقق من «طول» (length) مجموعة التغليف. يمكنك فعل ذلك عبر خاصية `length` التابعة للمصفوفات. إذا لم تُعِد الخاصية `length` القيمة `0`، فستعلم أنَّ هنالك عنصرًا واحدًا على الأقل قد طابق التعبير الذي مرَّرتَه إلى دالة jQuery. سنتحقق في الشيفرة الآتية من وجود عنصر له خاصية `id` ذات القيمة `notHere` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج: "0"
        alert($('#notHere').length);
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة

قد لا يخطر ببالك أنَّ خاصية `length` ستُعيد أيضًا عدد العناصر الموجودة في مجموعة التغليف. بتعبيرٍ آخر: عدد العناصر التي حُدِّدَت عبر التعبير الذي مُرِّرَ إلى دالة jQuery.

6. إنشاء مُرشِّحات مُخصَّصة لتحديد العناصر

من الممكن توسعة قدرات محرِّك التحديد في jQuery عبر إنشاء مُرشِّحات مُخصَّصة. نظريًا، ما نفعله هنا هو البناء على المُحدِّدات الموجودة في jQuery. لنقل مثلًا أننا نريد تحديد جميع العناصر في صفحة ويب التي مكانها «مطلق» (absolutely positioned). ولعدم امتلاك jQuery على مُرشِّحٍ مخصص باسم `positionAbsolute:` فعلينا إنشاء واحد بأنفسنا (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div style="position:absolute">absolute</div>
    <span style="position:absolute">absolute</span>
    <div>static</div>
    <div style="position:absolute">absolute</div>
    <div>static</div>
    <span style="position:absolute">absolute</span>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // $.expr[':'] تعريف مُرشِّح مخصص عبر توسعة
        $.expr[':'].positionAbsolute = function(element)
        {
          return $(element).css('position') === 'absolute';
        };

        // ما هو عدد العناصر ذات المكان المطلق في الصفحة؟
        alert($(':positionAbsolute').length); // الناتج: "4"

        // ما هو عدد عناصر <div> ذات المكان المطلق؟
        // الناتج: 2
```

```
        alert($('div:positionAbsolute').length);
      })(jQuery);
    </script>
  </body>
</html>
```

أهم شـيء عليـك أن تفهمـه هنـا هـو أنَّك لسـتَ مقيـدًا بالمُحـدِّدات الافتراضيـة الـتي توفرها jQuery. يمكنـك إنشـاء مُحـدِّدات خاصـة بـك؛ لكن قبـل أن تضيّع وقتـك بإنشـاء مُحـدِّد مُخصَّص فجـرب أولًا اسـتعمال الدالـة `filter()` مع تحديـد دالـة للترشـيح. على سـبيل المثـال، يمكنـك أن تتفادى كتابة المُرشِّح `:positionAbsolute` السابق بترشيح العناصر عبر تمرير دالة إلى الدالة `.filter()`.

```
// حذف جميع عناصر <div> من مجموعة التغليف //
// والتي ليس موقعها مطلقًا //
$('div').filter(function(){return $(this).css('position') ===
'absolute';});

// أو //

// حذف جميع العناصر من مجموعة التغليف //
// والتي ليس موقعها مطلقًا //
$('*').filter(function(){return $(this).css('position') ===
'absolute';});
```

ملاحظة	لمزيدٍ من المعلومات حول إنشاء مُحدِّدات مُخصَّصة، أقترح أن تقرأ هذه المقالة.

7. الفروقات بين الترشيح عبر الترتيب الرقمي والترشيح عبر العلاقات في شجرة DOM

توفِّر jQuery مُرشِّحاتٍ لترشيح مجموعة التغليف عبر الترتيب الرقمي لورود العناصر ضمن المجموعة، هذه المرشحات هي:

- `:first`
- `:last`
- `:even`
- `:odd`
- `:eq(index)`
- `:gt(index)`
- `:lt(index)`

ملاحظة	المُرشِّحات التي تُرشِّح مجموعة التغليف نفسها تبدأ بترشيح المجموعة من الرقم `0` (أو الفهرس `0`). على سبيل المثال، باستخدام `:eq(0)` و `:first` سنصل إلى العنصر الأول في المجموعة الذي هو في الفهرس `0` (مثلاً: `$('div:eq(0)')`)، وهذا ما يناقض تمامًا المرشِّح `:nth-child` الذي يبدأ من 1. أي على سبيل المثال `:nth-child(1)` سيعيد أول عنصر، ولن تستطيع استخدام `:nth-child(0)` حيث سيؤدي استخدام `:nth-child(0)` إلى عدم تحديد أيّة عناصر.

استـخدام `:first` سـيؤدي إلى تحديـد أول عنصـر في المجموعـة بينمـا اسـتخدام `:last` سيُحدِّد آخر عنصر في المجموعة. من المهم أن تتذكَّر أنَّ المُرشِّحات ستُرشِّح العناصر بناءً على مكانها (تبدأ الهيكلة الرقمية من الرقم `0`) ضمن المجموعة، ولا ترشِّحها بناءً على علاقتها مع بقية العناصـر الموجـودة في شـجرة DOM. وبالتـالي يجب أن تسـتنتج لمـاذا سـتُعيد المرشـحات `:first` و `:last` و `:eq(index)` عنصرًا وحيدًا فقط.

إذا لم يكن ذلـك واضـحًا بالنسـبة لـك، فاسـمح لي أن أشـرحها لـك أكثر. السـبب وراء إعـادة `:first` لعنصرٍ وحيدٍ فقط هو أنَّه يمكن أن يتواجد عنصرٌ وحيدٌ فقط في أوّل المجموعة عندما تكون هنالك مجموعةٌ وحيدة. يجب أن يكون الأمر منطقيًا بالنسبة لك، وتفحّص الشيفرة الآتيـة لتشاهد تطبيقًا عمليًا على هذا المفهوم (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
    </ul>
    <ul>
      <li>6</li>
      <li>7</li>
```

```
      <li>8</li>
      <li>9</li>
      <li>10</li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // تذكّر أنّ text() تُجمِّع محتوى جميع العناصر
        // الموجودة في مجموعة التغليف في سلسلةٍ نصيةٍ وحيدة
        alert('there are ' + $('li').length +
          ' elements in the set');
        // الحصول على أول عنصر في المجموعة
        alert($('li:first').text()); // الناتج: "1"
        // الحصول على آخر عنصر في المجموعة
        alert($('li:last').text()); // الناتج: "10"
        // إظهار العنصر السادس، تذكر أنّ العد يبدأ من 0
        alert($('li:eq(5)').text()); // الناتج: "6"
      })(jQuery);
    </script>
  </body>
</html>
```

بعد فهمك لكيفية ترشيح العناصر بناءً على مكانها، فيمكننا الآن الانتقال إلى ترشيح العناصر ذات العلاقة الفريدة بغيرها من العناصر ضمن شجرة DOM. توفّر jQuery عدّة مُحدّداتٍ لفعل ذلك. بعض تلك المُحدّدات خاصةٌ بمكتبة jQuery والأخرى شبيهةٌ بتعابير CSS الشهيرة لتحديد عناصر DOM.

- `ancestor descend`
- `parent > child`
- `prev + next`
- `prev ~ siblings`
- `:nth-child(selector)`
- `:first-child`
- `:last-child`
- `:only-child`
- `:empty`
- `:has(selector)`
- `:parent`

استخدام هذه المُحدّدات سيُحدّد العناصر بناءً على علاقتها مع بقية العناصر في شجرة DOM.

انظر مليًا إلى المثال الآتي لإزالة الغموض عمّا سبق (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
    </ul>
```

```
    <ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // تذكّر أنّ الدالة text() تجمع محتويات كل العناصر
        // الموجودة في مجموعة التغليف في سلسلةٍ نصيةٍ وحيدة

        // الناتج: "22"
        alert($('li:nth-child(2)').text());

        // الناتج: "135135"
        alert($('li:nth-child(odd)').text());

        // الناتج: "2424"
        alert($('li:nth-child(even)').text());
```

```
      // "2424" :الناتج
      alert($('li:nth-child(2n)').text());
    })(jQuery);
  </script>
</body>
</html>
```

إذا كنتَ مُستعجِبًا أنَّ `$('li:nth-child(odd)').text()` قد أعاد القيمة 135135 فأنت لم تفهم كيفية عمل المُرشِّحات. العبارة `$('li:nth-child(odd)')` تعني العثور على جميع عناصر `<li>` في صفحة الويب والتي هي «أبناء» (children) لعناصر أخرى ومن ثم ترشيحها والإبقاء على الأبناء التي ترتيبها فردي. عليك أن تنتبه أنَّ هنالك بنيتان في الصفحة تحتويان على عناصر `<li>` أبناء. الفكرة هي أنَّ مجموعة التغليف هنا مبنيةٌ على مُرشِّحٍ الذي يأخذ بالحسبان علاقة العنصر مع غيره من العناصر في DOM، وقد تتواجد مثل هكذا علاقة في أكثر من مكان في المستند.

المفهوم الذي عليك أن تعيه هنا هو أنَّ المُرشِّحات غير متساوية. فهنالك مُرشِّحات مبنية على علاقة العنصر مع غيره من العناصر (مثلًا: `:only-child`) وأخرى تُرشِّح العناصر بناءً على مكانها في مجموعة التغليف (مثلًا: `:eq()`).

8. تحديد العناصر عبر خاصية id عندما تحتوي قيمتها على محارف خاصة

تَستعمِل مُحدِّدات jQuery مجموعةً من المحارف الخاصة (مثلًا: `# ~ [] = >`) والتي يجب «تهريبها» (escape) عندما تُستعمَل كجزء من قيمة المُحدِّد (مثلًا: `id="#foo[bar]"`). من الممكن تهريب (escape) المحارف بوضع شرطتين مائلتين خلفيتين قبل المحرف. انظر إلى الشيفرة الآتية لترى كيف تمكنا من تحديدِ عنصرٍ له الخاصية `id` قيمتها `$foo[bar]` باستعمالنا لشرطتين مائلتين خلفيتين (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="#foo[bar]">jQuery</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        alert($('#\\#foo\\[bar\\]').text()); // "jQuery"
      })(jQuery);
    </script>
  </body>
</html>
```

هذه قائمة بكامل المحارف التي يجب تهريبها عندما تريد جعلها جزءًا من المُحدِّد:

- #
- ;
- &
- ,
- .
- +
- *
- ~
- '
- :
- "
- !
- ^
- $
-]
- [
-)
- (
- =
- <
- |
- /

9. إنشاء سلسلة من المُرشِّحات

من الممكن تجميع المُرشِّحات وراء بعضها، مثلًا: `a[title="jQuery"][href^="http://"][class!=""]`. يمكن توظيف هذه الميزة لتحديد عنصر يملك خاصيات معيّنة ذات قيم مُحدَّدة. على سبيل المثال، شيفرة jQuery الآتية ستُحدِّد عناصر `<a>` في صفحة HTML التي:

- تحتوي خاصية `href` تبدأ بالقيمة «`http://`»
- وتملك الخاصية `title` ذات القيمة «`jQuery`»
- وتملك الخاصية `class`

سيُحدَّد عنصر `<a>` وحيدٌ فقط (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a title="jQuery">jQuery.com</a>
    <a href="http://www.jquery.com" title="jQuery"
      class="foo">jQuery.com</a>
    <a href="">jQuery.com</a>
    <a href="http://www.jquery.com"
      title="jQuery">jQuery.com</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج: "1"
        alert(
          $('a[title="jQuery"][href^="http://"][class!=""]')
          .length);
      })(jQuery);
    </script>
  </body>
</html>
```

لاحظ كيف وضعنا ثلاثة مرشِّحات إلى جوار بعضها لكي نستطيع تحديد هذا العنصر.

من الممكن أيضًا وضع أنواع أخرى من المرشحات مع مرشحات الخاصيات. على سبيل المثال:

```
// تحديد آخر عنصر <div> موجود في مجموعة التغليف
// والذي يحتوي على السلسلة النصية jQuery
$('div:last:contains("jQuery")')

// تحديد جميع مربعات الاختيار الظاهرة والمُختارة
$(':checkbox:visible:checked')
```

الفكرة التي عليك أن تفهمها هنا هي أنَّك تستطيع استخدام أكثر من مُرشِّح معًا.

10. إنشاء مُرشِّحات متشعبة

من الممكن أيضًا إنشاء مُرشِّحات متشعبة، وهذا يسمح لنا باستعمال المرشحات بفعالية.

سأعطيك مثالًا عن كيفية تشعّب المرشحات لإجراء عمليات ترشيح معقدة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>javascript</div>
    <div><span class="jQuery">jQuery</span></div>
    <div>javascript</div>
    <div><span class="jQuery">jQuery</span></div>
    <div>javascript</div>
    <div><span class="jQuery">jQuery</span></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
```

```
    <script>
      (function($)
      {
        // تحديد جميع عناصر <div> وإزالة
        // جميع العناصر التي لها class="jQuery"

        // الناتج هو سلسلة نصية ناتجة عن دمج
        // محتوى كل العناصر المُحدّدة
        alert($('div:not(:has(.jQuery))').text());

        // تحديد جميع عناصر <div> وإزالة جميع
        // العناصر التي ليس ترتيبها فرديًا (العد يبدأ من 0)

        // الناتج هو سلسلة نصية ناتجة عن دمج
        // محتوى كل العناصر المُحدّدة
        alert($('div:not(:odd)').text());
      })(jQuery);
    </script>
  </body>
</html>
```

الفكرة الأساسية هنا هي أنَّك تستطيع إنشاء مُرشِّحات متشعبة.

ملاحظة

يمكنك أيضًا إنشاء مُرشِّحات متشعبة عبر الدالة `filter`، مثلًا:
`$('p').filter(':not(:first):not(:last)')`.

11. التعرف على استعمالات المُرشح ()nth-child:

المرشِّـح `:nth-child()` لـه العديـد من الاسـتعمالات، فيمكنـك مثلًا أن تُحـدِّد عـبره العنصر `<li>` الـذي ترتيبـه هـو الثـالث والموجـود ضـمن عنصر `<ul>`. تفحص الشـيفرة الآتيـة لكي تفهم كيفية استخدام المُرشح `:nth-child()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
      <li>6</li>
      <li>7</li>
      <li>8</li>
      <li>9</li>
      <li>10</li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
```

```
  (function($)
  {
    // تذكر أنّ الدالة text() تجمع محتويات جميع العناصر
    // الموجودة في مجموعة التغليف في سلسلة نصية واحدة

    // عبر الفهرس
    // الناتج: "1"
    alert($('li:nth-child(1)').text());

    // التي ترتيبها زوجي
    // الناتج: "246810"
    alert($('li:nth-child(even)').text());

    // التي ترتيبها فردي
    // الناتج: "13579"
    alert($('li:nth-child(odd)').text());

    // عبر معادلة رياضية
    // الناتج: "369"
    alert($('li:nth-child(3n)').text());

    // تذكر أنّ هذا المُرشح يبدأ العد فيه من 1
    // لا يوجد ناتج، لعدم وجود الفهرس 0
    alert($('li:nth-child(0)').text());
  })(jQuery);
</script>
```

```
  </body>
</html>
```

12. تحديد العناصر عبر البحث في قيم الخاصيات باستخدام التعابير النمطية

عندما تجد أنَّ مُرشِّحات الخاصيات في jQuery لتحديد العناصر غيرُ كافيةٍ بالنسبة لك، فجرِّب حينها استخدام التعابير النمطية (regular expressions). كَتَبَ James Padolsey إضافةً إلى مُحدِّدات المُرشحات للسماح لنا بإنشاء تعابير نمطية مخصصة لترشيح العناصر. وضعتُ مثالًا عنها هنا، لكن انظر أيضًا في james.padolsey.com للتفاصيل (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="123"></div>
    <div id="oneTwoThree"></div>
    <div id="0"></div>
    <div id="zero">
  </body>
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
  <script>
```

```
    (function($)
    {
      //James Padolsey filter extension
      jQuery.expr[':'].regex = function(elem, index, match)
        {
          var matchParams = match[3].split(','),
            validLabels = /^(data|css):/,
            attr = {
              method: matchParams[0].match(validLabels) ?
                matchParams[0].split(':')[0] : 'attr',
              property:
matchParams.shift().replace(validLabels, '')
            },
            regexFlags = 'ig',
            regex = new
RegExp(matchParams.join('').replace(/^\s+|\s+$/g, ''),
regexFlags);
          return regex.test(jQuery(elem)[attr.method]
(attr.property));
        }

      // التي <div> تحديد جميع عناصر
      // id تتواجد فيها أرقام في خاصية
      alert($('div:regex(id,[0-9])').length); // الناتج: "2"

      // تحديد جميع عناصر <div> التي تتواجد فيها
```

```
    // السلسلة النصية Two في خاصية id
    alert($('div:regex(id, Two)').length); // الناتج: "1"
  })(jQuery);

  </script>
  </body>
</html>
```

13. الفرق بين تحديد الأولاد المباشرين وبين تحديد الأبناء والأحفاد

يمكن تحديد الأبناء المباشرين فقط (direct children) باستخدام الرمز > أو عبر استخدام الدالة `children()`. أما جميع الأحفاد (أو العناصر «السليلة» [descendants]) يمكن أن تُحدَّد عبر استخدام التعبير *. تأكَّد أنَّك تفهم الفرق بينهما تمامًا. المثال الآتي يوضِّح ما سبق (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>
      <p><strong><span>text</span></strong></p>
      <p><strong><span>text</span></strong></p>
    </div>
    <script
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // كل تعبير برمجي سيُنتِج "2" لأنّ هنالك ابنين اثنين
        // مباشرَين هما العنصرين <p> داخل <div>
        alert($('div').children().length);

        // أو
        // alert($('>*', 'div').length);
        // alert($('div').find('>*').length);

        // كل تعبير برمجي سيُنتِج "6" لأنّ هنالك ستة أحفاد
        // (أي أبناء مباشرين وغير مباشرين)
        // دون احتساب العقد النصية (text node)
        alert($('div').find('*').length);

        // أو
        // alert($('*', 'div').length);
      })(jQuery);
    </script>
  </body>
</html>
```

14. تحديد الأولاد المباشرين عندما يُحدَّد السياق

من الممكن استخدام الرمز > دون سياقٍ (context) يسبقه لتحديد الأولاد المباشرين عندما نوفِّر السياق مسبقًا. أعلمُ أنَّ الكلام السابق غامض، لذا أنصحك بتفحص الشيفرة الآتية لفهم ما أقصد (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <ul id="firstUL">
      <li>text</li>
      <li>
        <ul id="secondUL">
          <li>text</li>
          <li>text</li>
        </ul>
      </li>
      <li>text</li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
```

```
        // تحديد عناصر <li> التي تُمثِّل
        // أبناءً مباشرةً لسياق التحديد
        // "3" الناتج
        alert($('ul:first').find('> li').length);

        // كان يمكن في السابق استخدام هذا التعبير
        // لكنه أصبح مهملًا
        // alert($('> li', 'ul:first').length);
      })(jQuery);
    </script>
  </body>
</html>
```

الفكرة الأساسية هي أنَّه يمكن أن يُستخدَم `'> element'` كتعبير عندما نُحدِّد سياق البحث.

الفصل الثالث:

التنقل في شجرة DOM



1. الفرق بين الدالتين ()find و ()filter

تُستخدَم الدالة `filter()` لترشيح المجموعة الحالية من العناصر الموجودة ضمن مجموعة التغليف. ويجب أن تستعملها عندما تريد ترشيح مجموعة من العناصر تم تحديدها سابقًا. على سبيل المثال، الشيفرة الآتية ستُرشِّح عناصر <p> الثلاثة الموجودة ضمن مجموعة التغليف (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p><strong>first</strong></p>
    <p>middle</p>
    <p><strong>last</strong></p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج middle، وذلك بعد ترشيح أول وآخر عنصر
        // <p> موجود ضمن مجموعة التغليف
        alert(
          $('p').filter(':not(:first):not(:last)').text()
        );
```

```
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة

عند استخدامك للدالة `filter()` فاسأل نفسك إن كان استخدامها ضروريًا فعلًا. على سبيل المثال يمكن إعادة كتابة العبارة البرمجية `$('p').filter(':not(:first):not(:last)')` دون استخدام الدالة `filter()` بتمرير مُحدِّدات خاصة كتعبير تحديد للدالة `jQuery` كالتالي `$('p:not(:first):not(:last)')`.

أما الدالة `find()` فستُستخدَم للعثور على «أحفاد» (descendants) العناصر المُحدَّدة حاليًا.

تخيل أنَّ الدالة `find()` ستؤدي إلى تحديث أو تغيير مجموعة التغليف الحالية لكي تحتوي على عناصرَ جديدةٍ كانت متواجدةً ضمن العناصر المُحدَّدة مسبقًا. على سبيل المثال، الشيفرة الآتية ستؤدي إلى تغيير مجموعة التغليف من عناصر `<p>` إلى عنصرَي `<strong>` عبر استخدام الدالة `find()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p><strong>first</strong></p>
    <p>middle</p>
    <p><strong>last</strong></p>
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج: "strong"
        alert($('p').find('strong').get(0).nodeName);
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة

يمكنك في الواقع أن تدمج العناصر الموجودة مسبقًا في مجموعة التغليف مع العناصر الجديدة المُحدَّدة عبر استخدام الدالة `find()` وذلك باستخدام الدالة `addBack()`، مثال: `$('p').find('strong').addBack()`.

الفكرة هنا هي أنَّ الدالة `filter()` تؤدي إلى تقليل عدد (أو ترشيح) العناصر المُحدَّدة حاليًا في مجموعة التغليف بينما الدالة `find()` ستُنشِئ مجموعة تغليف جديدة.

ملاحظة

الدالتان `find()` و `filter()` هما دالتان هادمتان (destructive) واللتان يمكن إبطال تأثيرهما باستخدام `end()` والتي ستُعيد مجموعة التغليف إلى حالتها السابقة قبل استخدام `find()` أو `filter()`.

2. تمرير دالة إلى ()filter بدلًا من تعبير

قبل أن تجري مسرعًا وتُنشِئ مُرشِّحًا مخصصًا لتحديد العناصر، فمن المنطقي والأسهل أن تُمرِّر دالةً إلى `filter()` والتي ستسمح لك باختبار تحقيق شرط معيّن على كل عنصر موجود في مجموعة التغليف.

على سبيل المثال لنقل أننا نريد وضع جميع عناصر `<img>` في صفحة HTML ضمن عنصر `<p>` في حال لم يكن عنصر `<img>` موجودًا من البداية ضمن عنصر `<p>`.

يمكنك إنشاء مُرشِّح مخصص لإتمام هذه المهمة أو يمكنك استخدام الدالة `filter()` بتمرير دالة إليها والتي ستُحدِّد إن كان العنصر الأب (parent) هو العنصر `<p>`، فإذا تحقق ذلك فسيُزال هذا العنصر من المجموعة، ومن ثم ستوضع جميع عناصر `<img>` المتبقية في مجموعة التغليف ضمن عناصر `<p>`.

كتبتُ الشيفرة التالية والتي ستُحدِّد كل عناصر `<img>` في صفحة HTML، ومن ثم سأمرر دالةً إلى `filter()` والتي ستُستخدَم للمرور على كل عنصر من العناصر الموجودة في مجموعة التغليف (باستخدام `this`) وستختبر إذا كان العنصر الأب لكل عنصر `<img>` هو `<p>` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <img />
    <img />
```

```
    <p><img /></p>
    <img />
    <p><img /></p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('img').attr('src',
'http://static.jquery.com/files/rocker/images/logo_jquery_215
x53.gif')
          .filter(
          function(){return !$(this).parent('p').length == 1}
          ).wrap('<p></p>');
      })(jQuery);
    </script>
  </body>
</html>
```

لاحظ كيف استعملتُ المعامل ! لتغيير القيمة المنطقية من true إلى false. وهذا لأنني أريد حذف عناصر <img> والتي يكون العنصر <p> هو العنصر الأب لها من المجموعة. الدالة التي مُرِّرتُها إلى ()filter ستحذف العناصر من المجموعة إن أعادت الدالةُ القيمةَ false.

الفكرة الأساسية هنا هي إذا كنتَ تتعامل مع حالةٍ وحيدةٍ فقط، فإنشاء مُرشِّحٍ مخصصٍ (مثلًا: findImgWithNoP:) لتطبيقه على حالةٍ وحيدةٍ قد يكون مضيعةً للوقت، وتستطيع تحقيق

الهدف نفسه بتمرير دالة إلى `filter()`. الطريقة التي شرحناها هنا هي طريقةٌ مفيدةٌ جدًا، تخيّل ما نستطيع فعله عندما نستخدم التعابير النمطية (regular expressions) في دالة `filter()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <ul>
      <li>jQuery is great.</li>
      <li>Its lightweight.</li>
      <li>Its free!</li>
      <li>jQuery makes everything simple</li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // وضع عنصر <strong> حول أيّ نص موجود ضمن
        // عنصر <li> والذي يحتوي على الكلمة jQuery
        var pattern = /jQuery/i;
        $('ul li').filter(function()
        {
          return pattern.test($(this).text());
```

```
        }).wrap('<strong></strong>');
      })(jQuery);
    </script>
  </body>
</html>
```

3. التنقل في شجرة DOM

يمكنــك ببســاطة «الانتقــال إلى الأعلى» في شــجرة DOM باســتخدام الــدوال parent() و parents() و closest(). من المهم أن تفهم الفروقــات بين هــذه الــدوال. تفحص الشــيفرة الآتية لكي تفهم ما هو الفرق بين دوال التنقل في jQuery (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="parent2">
      <div id="parent1">
        <div id="parent0">
          <div id="start"></div>
        </div>
      </div>
      <div>
        <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
```

```
<script>
  (function($)
  {
    // الناتج: "parent0" x4
    alert($('#start').parent().attr('id'));
    alert(
      $('#start').parents('#parent0').attr('id'));
    alert($('#start').parents()[0].id);
    alert(
      $('#start').closest('#parent0').attr('id'));

    // الناتج: "parent1" x4
    alert($('#start').parent().parent().attr('id'));
    alert(
      $('#start').parents('#parent1').attr('id'));
    alert($('#start').parents()[1].id);
    alert(
      $('#start').closest('#parent1').attr('id'));

    // الناتج: "parent2" x4
    alert(
      $('#start').parent().parent().parent()
      .attr('id')
    );
    alert(
      $('#start').parents('#parent2').attr('id'));
```

```
            alert($('#start').parents()[2].id);
            alert(
              $('#start').closest('#parent2').attr('id'));

          })(jQuery);
        </script>
  </body>
</html>
```

ملاحظات

- ربما يبدو لك أنَّ `closest()` و `parents()` لهما نفس الوظيفة، إلا أنَّ الدالة `closest()` ستُضمِّن العنصر الحالي المُحدَّد في عملية الترشيح (أي تستطيع تحديد العنصر الحالي فيها لو نفَّذتَ `$('#start').closest('#start')`، لكنك لا تستطيع فعل ذلك مع `parents()`.

- تتوقف `closest()` عن البحث عندما تجد مُطابقةً، بينما `parents()` فستحصل على جميع العناصر «الآباء» ثم تُرشِّحها بناءً على التعبير الذي مرَّرتَه إليها. وبالتالي لن تتمكن الدالة `closest()` من إعادة أكثر من عنصرٍ وحيدٍ فقط.

4. دوال التنقل تقبل تمرير تعبيرات CSS كوسائط اختيارية

لا تُمرَّر تعابير CSS إلى دالة jQuery لتحديد العناصر فحسب، وإنما يمكن أيضًا تمريرها إلى عدّة دوال للتنقل في شجرة DOM. ربما يسهل علينا نسيان ذلك لوجود عدد كبير من دوال التنقل دون أن نحتاج إلى استخدام التعابير مطلقًا (مثلًا: `next()`).

تمرير التعبيرات إلى دوال التنقل هو أمرٌ اختياريٌ، لكن تذكَّر أنَّك تستطيع فعل ذلك.

- `children('expression')`
- `next('expression')`
- `nextAll('expression')`
- `parent('expression')`
- `parents('expression')`
- `prev('expression')`
- `prevAll('expression')`
- `siblings('expression')`
- `closest('expression')`

الفصل الرابع:

تعديل شيفرات HTML



1. إنشاء وإضافة وتعديل HTML أثناء التنفيذ

يمكنك إنشاء شيفرات HTML أثناء التنفيذ عبر تمرير سلسلة نصية فيها شيفرة HTML إلى دالة jQuery (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // "DIV" :الناتج
        alert($('<div><a></a></div>').get(0).nodeName);
        // الناتج: "1"، أي هنالك عنصر div وحيد
        alert($('<div><a></a></div>').length);
        // الناتج: "2"، أي هنالك عنصرَي div
        alert(
          $('<div><a></a></div><div><a></a></div>').length);
      })(jQuery);
    </script>
  </body>
</html>
```

من المهم أن تلاحظ أنّه عند إنشاء بُنى DOM باستخدام دالة jQuery، فستُضاف العناصر الرئيسية فقط إلى مجموعة التغليف. فمثلًا في الشيفرة السابقة، ستتواجد عناصر <div> فقط في مجموعة التغليف.

يمكنك استخدام الدالة find() للتعامل مع أيّة عناصر في بنية HTML بعد إنشائها.

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('<div><a></a></div>').find('a').text('jQuery')
          .attr('href', 'http://www.jquery.com');
      })(jQuery);
    </script>
  </body>
</html>
```

بعد الانتهاء من التعامل مع شيفرة HTML الجديدة، فسنتمكن من إضافتها إلى شجرة DOM عبر استخدام إحدى دوال jQuery. سنستخدم في المثال الآتي الدالة appendTo() لإضافة الشيفرة إلى الصفحة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('<div><a></a></div>')
          .find('a')
          .text('jQuery')
          .attr('href', 'http://www.jquery.com')
          .end().appendTo('body');
          // find() للخروج من الدالة end() استخدمنا
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظات	- العناصر البسيطة التي لا تحتوي على خاصيات (مثلًا: `$('<div></div>')`) تُنشَأ عبر الدالة `document.createElement` بينما ستُستعمَل الخاصية `innerHTML` في بقية الحالات. وفي الواقع، تستطيع تمرير عنصر مُنشَأ عبر `document.createElement` إلى دالة jQuery (مثال: `$(document.createElement('div'))`).

- سلسلة HTML النصية المُمرِّرة إلى دالة jQuery لا يمكن أن تحتوي على عناصر لا يُسمَح بأن يحتويها العنصر <div>.

- سلسلة HTML النصية المُمرِّرة إلى دالة jQuery يجب أن تكون مكتوبةً بطريقةٍ سليمةٍ تمامًا.

- يجب أن تُفتَح وتُغلَق جميع عناصر HTML عندما تُمرِّر سلسلةً نصيةً فيها HTML إلى دالة jQuery. إن لم تفعل ذلك فقد تتسبب بحدوث علل وخصوصًا في متصفح IE. أغلق جميع عناصر HTML دومًا ولا تستخدم الوسوم المختصرة (مثلًا: <div />).

2. فهم آلية عمل الدالة ()index

من الممكن تحديد فهرس أحد العناصر داخل مجموعة التغليف بتمرير ذاك العنصر إلى الدالة ()index. افترض مثلًا أنَّ لديك مجموعة تغليف تحتوي على جميع عناصر <div> الموجودة في صفحة ويب وتريد أن تعرف فهرس آخر عنصر <div> فيها (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>0</div>
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <script
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج: "3"
        alert($('div').index($('div:last')));

        // تغيّرت آلية عمل index() قليلًا في إصدار 1.3.3
        // أصبح الآن بإمكاننا استخدام هذه الشيفرة
        // $('div:last').index()
        // لإعادة القيمة 3 في هذا المثال.
      })(jQuery);
    </script>
  </body>
</html>
```

لن تفهم استخدام index() تمامًا حتى ترى كيف يمكن استخدامها مع الأحداث (events).

فمثلًا، عند النقر على عناصر <div> الموجودة في الصفحة فستُمرِّر العنصر <div> الذي تم النقر عليه (باستخدام الكلمة المحجوزة this) إلى الدالة index() لتحديد فهرسه (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="nav">
```

```
      <div>nav text</div>
      <div>nav text</div>
      <div>nav text</div>
      <div>nav text</div>
      <div>
        <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
        <script>
          (function($)
          {
            // إظهار ترتيب عنصر div الذي ضُغِطَ عليه من بين
            // جميع عناصر div الموجودة في مجموعة التغليف
            $('#nav div').click(function()
            {
              alert($('#nav div').index(this));

              // أو يمكنك استخدام هذه الخدعة الجميلة
              // alert($(this).prevAll().length);
            });
          })(jQuery);
        </script>
  </body>
</html>
```

3. فهم طريقة عمل الدالة text()

قد يفترض أحدنا مُخطِئًا أنَّ الدالة `text()` ستُعيد العقدة النصية (text node) لأول عنصر في مجموعة التغليف؛ وإنما ستجمع محتوى جميع العقد النصية لجميع العناصر الموجودة في مجموعة التغليف وتُعيد القيمة النهائية كسلسلةٍ نصيةٍ وحيدة. تأكَّد أنَّك قد فهمتَ هذه الفكرة وإلا فقد تحصل على نتائج غير متوقعة عند استخدامها (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>1,</div>
    <div>2,</div>
    <div>3,</div>
    <div>4</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        alert($('div').text()); // الناتج: "1,2,3,4"
      })(jQuery);
    </script>
  </body>
</html>
```

4. تبديل أو إزالة المحارف باستخدام التعابير النمطية

باستخدامنا للدالة `replace()` الموجودة في JavaScript –مع استعمال دوال مكتبة jQuery– سنتمكّن بكل سهولة من تبديل أو حذف أيٍّ نمطٍ من المحارف موجودٍ في النص المحتوى ضمن أحد العناصر (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p>
      I really hate using JavaScript.
      I mean really hate it!
      It is the best crap ever!
    </p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {

        var $p = $('p');
        // hate بدلًا من love وضع
        $p.text($p.text().replace(/hate/ig, 'love'));
```

```
      // حذف crap وعدم وضع أي شيء مكانها
      $p.text($p.text().replace(/crap/ig, ''));

      // أبقِ في بالك أنّ الدالة replace() تُعيد سلسلةً نصيةً
      // ولا تعيد كائن jQuery، وهذا هو السبب وراء
      // وضع الدالة replace() بعد الدالة text()
    })(jQuery);
  </script>
 </body>
</html>
```

يمكنك أيضًا تحديث أيّة محارف موجودة ضمن سلسلة نصية مُعادة من الدالة `html()`، وهذا يعني أنّك تستطيع تحديث النص، وتستطيع أيضًا تحديث واستبدال عناصر DOM عبر التعابير النمطية.

5. شرح كيفية عمل الدالة contents()

يمكن أن تُستخدَم الدالة `contents()`. للعثور على عقد (nodes) جميع العناصر الأبناء، بما في ذلك العقد النصية (text node) الموجودة داخل العنصر. لكن انتبه، إن كانت المحتويات الناتجة عن هذه الدالة هي عقد نصيّة فقط فسيوضع الناتج داخل مجموعة التغليف كعقدة نصية وحيدة. لكن إن كانت المحتويات تحتوي على عنصر أو أكثر بالإضافة إلى العقد النصية، فستُعيد الدالة `contents()`. العقد النصية وعقد العناصر معًا. امعن النظر في الشيفرة السابقة لتفهم الشرح السابق (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p>I love using jQuery!</p>
    <p>I love <strong>really</strong> using jQuery!</p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // HTML لعدم وجود عناصر "I love using jQuery!" ينتج
        alert($('p:first').contents().get(0).nodeValue);
        // "I love" :الناتج
        alert($('p:last').contents().get(0).nodeValue);
        // نصية عقدة وليس HTML لكنه عنصر "really" :الناتج
        alert($('p:last').contents().eq(1).text());
        // "using jQuery!" :الناتج
        alert($('p:last').contents().get(2).nodeValue);
      })(jQuery);
    </script>
  </body>
</html>
```

لاحظ أنَّه إذا كان العنصرُ في مجموعة التغليف عقدةً نصيةً، فعلينا استخراج المحتوى عبر `..get(0).nodeValue`

الدالة `contents()` مفيدةٌ عند استخراج قيم العقد النصية. ومن الممكن استخراج العقد النصية فقط من بُنية DOM عبر استخدام `contents()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p>jQuery gives me
      <strong>more <span>power</span></strong>
       than any other web tool!
    </p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('p')
          .find('*')  // تحديد جميع العقد
          .addBack()  // بما في ذلك <p>
          .contents() // الحصول على جميع العقد الأبناء بما
                      // في ذلك العقد النصية
          .filter(function()
          {
            return this.nodeType == Node.TEXT_NODE;
          }) // حذف جميع العقد غير النصية
```

```
          .each(function(i, text)
          {
            alert(text.nodeValue)
          }); // إظهار النصوص الموجودة ضمن مجموعة التغليف
      })(jQuery);
    </script>
  </body>
</html>
```

هنالك استخدامٌ آخر للدالة `contents()`، حيث تعطينا وصولًا إلى مستند HTML الموجود ضمن عنصر `<iframe>`، لطالما كانت الصفحة الموجودة ضمن عنصر `<iframe>` والصفحة «الأب» في نفس النطاق (domain):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <iframe src="iframe.html" height="100px"
width="100px"></iframe>
    <!-- iframe محتوى عنصر -->
    <!--
    <!DOCTYPE html>
    <html lang="en">
    <body>
    <body>
    <p>Hi, I am the content inside of the iframe's body
```

```
element.</p>
    </body>
    </html>
    -->
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {

        /* الناتج: "<p>Hi, I am the content inside of the
iframe's body element.</p>" */
        alert($('iframe').contents().find('body').html());
        // أو عبر تمرير «السياق» إلى الدالة jQuery ...
        /* الناتج: "<p>Hi, I am the content inside of the
iframe's body element.</p>" */
        alert($('body', $('iframe').contents()).html());

      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة	حاول الكثيرون استخدام `contents()` للوصول إلى محتوى عنصر `<iframe>` ليس موجودًا بنفس النطاق وواجهوا مشكلة «سياسة المصدر الواحد» (same-origin policy). تأكَّد أنَّك تفهم محدوديات استخدام JavaScript وعناصر `iframe` قبل أن تحاول الوصول إلى أو تعديل محتوى `iframe`. هنالك إضافةٌ (موجودةٌ في http://code.google.com/p/jquery-crossframe/) التي تعطيك حلًا التفافيًا إذا أردت التواصل مع `<iframe>` موجودٍ في نطاقٍ آخر.

6. استخدم remove() لن يُزيل العناصر من مجموعة التغليف

عندما تستخدم `remove()` لحذف جزء من DOM فإن العناصر الموجودة ضمن بُنية DOM المحذوفة ستبقى موجودةً ضمن مجموعة التغليف. يمكنك إذًا أن تحذف عنصرًا، ثم تجري تعديلات على ذاك العنصر، ثم تضعه مرةً أخرى في شجرة DOM، وذلك عبر سلسلة jQuery وحيدة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>remove me</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
```

```
      {
        $('div').remove().html('<a href=
          "http://www.jQuery.com">jQuery</a>')
          .appendTo('body');
      })(jQuery);
    </script>
  </body>
</html>
```

الفكرة الأساسية هنا هي أنَّك إذا استخدمتَ `remove()` لحذف العناصر من شجرة DOM، فهذا لا يعني أنَّ تلك العناصر ستُحذَف من مجموعة التغليف في jQuery.

الفصل الخامس:

نماذج HTML



1. تفعيل وتعطيل عناصر النموذج

نستطيع بسهولة باستخدام jQuery تعطيل عناصر النماذج عبر ضبط الخاصية disabled لعناصر النموذج إلى القيمة disabled. يمكننا فعل ذلك بكل بساطة باختيار حقل إدخال، ثم استخدام الدالة attr() لضبط الخاصية disabled (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input name="button" type="button" id="button"
value="button" />
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('#button').attr('disabled', 'disabled');
      })(jQuery);
    </script>
  </body>
</html>
```

لتفعيـل حقـل مُعطّـل ، يمكننـا ببسـاطة حـذف الخاصـية disabled باسـتخدام الدالـة removeAttr() أو عـبر ضـبط قيمـة الخاصـية disabled إلى سلسـلةٍ نصـيةٍ فارغـة باسـتخدام attr() (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input name="button" type="button" id="button"
value='button' disabled="disabled" />
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('#button').removeAttr('disabled');

        // أو
        // $('#button').attr('disabled', '');
      })(jQuery);
    </script>
  </body>
</html>
```

2. كيفية تحديد إذا كان أحد عناصر النموذج مُفعَّلًا أم معطّلًا

باستخدام مُرشِّحات jQuery الخاصة بالنماذج `:disabled` أو `:enabled` يسهل علينا تحديد ومعرفة إذا كان عنصرٌ في نموذجٍ معطّلًا أم مفعّلًا بإعادة قيمة منطقية (boolean). انظر مليًّا إلى الشيفرة الآتية للتوضيح (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input name="button" type="button" id="button1"
value="button" />
    <input name="button" type="button" id="button2"
disabled="disabled" value="button" />
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // اختبار إن كان مُفعّلًا باستخدام is
        alert($('#button1').is(':enabled')); // الناتج: true

        // أو عبر استخدام مُرشِّح
        alert($('#button1:enabled').length); // الناتج: "1"
```

```
        // اختبار إن كان مُعطلًّا باستخدام is
        alert($('#button2').is(':disabled')); // "true"

        // أو عبر استخدام مُرشِّح
        alert($('#button2:disabled').length); // الناتج: "1"

      })(jQuery);
    </script>
  </body>
</html>
```

3. اختيار أو عدم اختيار مربع اختيار أو مربع انتقاء

يمكنك اختيار مربع انتقاء (radio) أو مربع اختيار (checkbox) عبر ضبط خاصية checked إلى القيمة true باستخدام الدالة attr() (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input name="" value="" type="checkbox" />
    <input name="" value="" type="radio" />
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
```

```
      (function($)
      {
        // اختيار جميع مربعات الاختيار أو الانتقاء
        $('input:checkbox,input:radio')
          .attr('checked', 'checked');
      })(jQuery);
    </script>
  </body>
</html>
```

لإلغاء اختيار مربع اختيار أو انتقاء، فاحذف الخاصية checked باستخدام الدالة removeAttr() أو عبر ضبط قيمة الخاصية checked إلى سلسلةٍ نصيةٍ فارغة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input name="" type="checkbox" value=""
checked="checked">
    <input name="" type="radio" value="" checked="checked">
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
```

```
        $('input').removeAttr('checked');
        // أو
        $('input').attr('checked', '');
      })(jQuery);
    </script>
  </body>
</html>
```

4. اختيار وإلغاء اختيار عدّة مربعات اختيار وانتقاء

يمكنك استخدام الدالة `val()` في jQuery لاختيار عدّة حقول اختيار وانتقاء. وذلك بتمرير مصفوفة تحتوي عدّة سلاسل نصية تتوافق قيمتها مع قيمة الخاصية `value` لحقول الاختيار والانتقاء (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input type="radio" value="radio1">
    <input type="radio" value="radio2">
    <input type="checkbox" value="checkbox1">
    <input type="checkbox" value="checkbox2">
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
```

```
      (function($)
      {
        // اختيار جميع حقول الاختيار والانتقاء في الصفحة
        $('input:radio,input:checkbox').val(['radio1',
'radio2', 'checkbox1', 'checkbox2']);

        // يمكنك استخدام ما يلي لإلغاء اختيار جميع العناصر
        // $('input:radio,input:checkbox')
        //    .removeAttr('checked');
        // أو
        // $('input:radio,input:checkbox')
        //    .attr('checked', '');
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة	إذا كان مربع اختيار أو انتقاء مُختارًا من قبل، فلن يؤدي استخدام `val()` إلى إلغاء اختياره.

5. معرفة فيما إذا كان مربع اختيار أو مربع انتقاء مختارًا أم لا

يمكنك معرفة إذا تم اختيار أو لم يتم اختيار مربع اختيار أو انتقاء عبر استخدام المُرشِّح `:checked`. انظر إلى الشيفرة الآتية لتشاهد عدّة استخدامات للمُرشِّح `:checked` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input checked="checked" type="checkbox" />
    <input checked="checked" type="radio" />
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج: "true"
        alert($('input:checkbox').is(':checked'));

        // أو إضافة العناصر إلى مجموعة التغليف إن
        // تم اختيارها. الناتج 1
        alert($('input:checkbox:checked').length);

        // الناتج: "true"
        alert($('input:radio').is(':checked'));

        // أو إضافة العناصر إلى مجموعة التغليف إن
        // تم اختيارها. الناتج 1
        alert($('input:radio:checked').length);
      })(jQuery);
```

```
    </script>
  </body>
</html>
```

6. معرفة إذا كان حقلٌ في النموذج مخفيًا

يمكنك معرفة إذا كان حقلٌ ما في النموذج مخفيًا (hidden) عبر المُرشِّح `:hidden`. انظر إلى الشيفرة الآتية لتعاين استخدامات مختلفة للمُرشِّح `:hidden` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input type="hidden" />
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج: "true"
        alert($('input').is(':hidden'));

        // أو إضافة العناصر المخفية إلى مجموعة التغليف
        // الناتج 1
        alert($('input:hidden').length);
```

```
      })(jQuery);
    </script>
  </body>
</html>
```

7. ضبط أو الحصول على قيمة الخاصية value لحقل إدخال

يمكن استخدام الدالة ‎`val()`‎ لضبط أو الحصول على قيمة الخاصية `value` لعنصر إدخال (أي عناصر `button` و `checkbox` و `hidden` و `image` و `password` و `radio` و `reset` و `submit` و `text` وغير ذلك من عناصر نماذج HTML5 ...). سأضبط في المثال الآتي قيمةَ كل حقلِ إدخالٍ باستخدام ‎`val()`‎ ثم سأعرضها عبر ‎`alert()`‎ و ‎`val()`‎ (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input type="button" />
    <input type="checkbox" />
    <input type="hidden" />
    <input type="image" />
    <input type="password" />
    <input type="radio" />
    <input type="reset" />
    <input type="submit" />
    <input type="text" />
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {

        $('input:button').val('I am a button')
        $('input:checkbox').val('I am a checkbox')
        $('input:hidden').val('I am a hidden input')
        $('input:image').val('I am a image')
        $('input:password').val('I am a password')
        $('input:radio').val('I am a radio')
        $('input:reset').val('I am a reset')
        $('input:submit').val('I am a submit')
        $('input:text').val('I am a text')

        // إظهار قيمة الخاصية value
        alert($('input:button').val());
        alert($('input:checkbox').val());
        alert($('input:hidden').val());
        alert($('input:image').val());
        alert($('input:password').val());
        alert($('input:radio').val());
        alert($('input:reset').val());
        alert($('input:submit').val());
```

```
        alert($('input:text').val());

      })(jQuery);
    </script>
  </body>
</html>
```

8. ضبط والحصول على القيمة المُحدَّدة لعنصر اختيار من متعدد

يمكننا تحديد القيم المُختارة باستخدام الدالة `val()` في عنصر اختيار من متعدد (multi-select) بتمرير مصفوفة إلى الدالة `val()` تحتوي على القيم النصية للخيارات الموافقة.

وللحصول على القيم المُختارة في عنصر اختيار من متعدد، نستطيع أيضًا استخدام الدالة `val()` للحصول على مصفوفة للقيم المختارة. ستحتوي المصفوفة على قيم الخاصية `value` لجميع القيم المُختارة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <select size="4" multiple="multiple">
      <option value="option1">option one</option>
      <option value="option2">option two</option>
      <option value="option3">option three</option>
      <option value="option4">option four</option>
    </select>
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // ضبط القيم المختارة
        $('select').val(['option2', 'option4']);

        // الحصول على القيم المُختارة
        // الناتج: "option2, option4"
        alert($('select').val().join(', '));
      })(jQuery);
    </script>
  </body>
</html>
```

9. ضبط والحصول على النص الموجود ضمن عنصر textarea

يمكنك ضبط محتوى العقدة النصية لعنصر <textarea> عبر تمرير سلسلة نصية إلى الدالة val() لتُستخدَم كمحتوى نصي للعنصر. ولكي نحصل على محتويات العنصر textarea فسنستخدم أيضًا الدالة val() كما هو ظاهر في المثال الآتي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
```

```
  <body>
    <textarea></textarea>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // ضبط محتوى مربع النص
        $('textarea').val('I am a textarea');

        // الناتج: "I am a textarea"
        alert($('textarea').val());
      })(jQuery);
    </script>
  </body>
</html>
```

10. ضبط والحصول على قيمة الخاصية value للعنصر button

يمكنك ضبط قيمة الخاصية value لعنصر button بتمرير سلسلة نصية إلى الدالة ()val.

ولكي نحصل على قيمة الخاصية value التابعة لعنصر <button> فيمكننا أن نستعمل الدالة ()val أيضًا (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <button>Button</button>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // ضبط العنصر إلى
        //<button value="I am a Button Element">
        $('button').val('I am a Button Element')

        // الناتج: "I am a Button Element"
        alert($('button').val());
      })(jQuery);
    </script>
  </body>
</html>
```

11. تعديل مكونات العنصر select

تُسهِّل jQuery كثيرًا من إجراء بعض المهام الشائعة عند تعديل العنصر <select>. سأشرح فيما يلي بعض تلك المهام وسأريك مثالًا عن الشيفرة.

- إضافة خيارات جديدة في النهاية:

```
$('select').append('<option value="">option</option>');
```

- إضافة خيارات جديدة إلى البداية:

```
$('select').prepend('<option value="">option</option>');
```

- وضع خيارات جديدة بدلًا من القديمة:

```
$('select').html('<option value="">option</option><option
value="">option</option>');
```

- استبدال عناصر ذات فهرس (index) معيّن؛ عبر استخدام المُرشِّح `eq()`: لتحديد العناصر ثم استخدام الدالة `.replaceWith()` لاستبدالها:

```
$('select option:eq(1)').replaceWith('<option
value="">option</option>');
```

- ضبط العنصر المُختار في `select` إلى العنصر ذي الفهرس 2:

```
$('select option:eq(2)').attr('selected', 'selected');
```

- إزالة آخر خيار من الخيارات:

```
$('select option:last').remove();
```

- تحديد خيارات معيّنة من عنصر `select` عبر ترتيبها في مجموعة التغليف باستخدام مُرشِّحات خاصة:

```
$('#select option:first');
$('#select option:last');
$('#select option:eq(3)');
$('#select option:gt(5)');
$('#select option:lt(3)');
$('#select option:not(':selected')');
```

- الحصول على القيم النصية لجميع الخيارات المُختارة:

```
$('select option:selected').text();
```

- الحصول على قيمة الخاصية `value` لأحد خيارات العنصر `select` (ستُحدِّد آخر خيار في مثالنا):

```
$('select option:last).val();
```

- الحصول على فهرس العنصر المُختار (الترقيم بدءًا من الصفر)؛ يجدر بالذكر أنَّ الشيفرة

الآتية لن تعمل مع العناصر التي يمكن اختيار أكثر من خيار فيها:

```
$('select option').index($('select option:selected'));
```

- إضافة خيار بعد مكانٍ معين:

```
$('select option:eq(1)').after('<option
value="">option</option>');
```

- إضافة خيار قبل مكانٍ معين:

```
$('select option:eq(3)').before('<option
value="">option</option>');
```

12. تحديد حقول النموذج عبر نوعها

من الممكن تحديد حقول النماذج عبر نوعها (مثلًا: `$('input:checkbox')`) توفِّر jQuery المُرشِّحات الآتية المرتبطة بأنواع حقول النموذج لتحديد العناصر حسب نوعها.

- `:text`
- `:password`
- `:radio`
- `:checkbox`
- `:submit`
- `:image`
- `:reset`
- `:file`
- `:button`

13. تحديد جميع عناصر النماذج

من الممكن تحديد جميع عناصر النماذج عبر استخدام المُرشِّح ‎:input‎. لن يُحدِّد هذا المُرشِّح عناصر input فحسب، وإنما سيُحدِّد أيّة عناصر <textarea> و <select> و <button> أيضًا. لاحظ قيمة length لمجموعة التغليف في المثال الآتي عند استخدام المُرشِّح ‎:input‎ (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input type="button" value="Input Button" />
    <input type="checkbox" />
    <input type="file" />
    <input type="hidden" />
    <input type="image" />
    <input type="password" />
    <input type="radio" />
    <input type="reset" />
    <input type="submit" />
    <input type="text" />
    <select>
      <option>Option</option>
    </select>
    <textarea></textarea>
    <button>Button</button>
    <script
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // الناتج هو 13 عنصرًا من عناصر النماذج
        alert($(':input').length);
      })(jQuery);
    </script>
  </body>
</html>
```

الفصل السادس:

الأحداث في jQuery



1. لسنا مقيّدين بحدث ready() واحد فقط

من المهم أن تبقي بذهنك أنَّه من الممكن استخدام الحدث المخصص `ready()` عَدَدَ ما شئتَ؛ ولستَ مقيدًا بإضافة حدث `ready()`. وحيد إلى المستند. وستُنفَّذ الشيفرات المرتبطة بالحدث `ready()` بنفس ترتيب ورودها.

> **ملاحظة**
>
> تمرير دالة إلى دالة jQuery (مثلًا:
> `jQuery(function(){//code here})`) هي طريقةٌ مختصرةٌ للحدث
> `ready()` (أي `jQuery(document).ready()`).

2. إضافة أو إزالة الأحداث باستخدام on() و off()

باستخدام الدالة `on()` (مثلًا: `jQuery('a').on('click', function(){})`) يمكننا إضافة أيٍّ من معالجات الأحداث القياسية (standard handlers) إلى عناصر DOM الملائمة.

- `blur`
- `focus`
- `load`
- `resize`
- `scroll`
- `unload`
- `beforeunload`
- `click`
- `dblclick`
- `mousedown`
- `mouseup`
- `mousemove`
- `mouseover`
- `mouseout`
- `change`
- `select`
- `submit`
- `keydown`

- keypress
- keyup
- error

كما هو واضح -واعتمادًا على معايير DOM- يمكن أن ترتبط معالجات أحداث مُحدَّدة بعناصر معيّنة.

بالإضافة إلى القائمة السابقة الخاصة بمعالجات الأحداث القياسية، يمكنك أيضًا استخدام الدالة `on()` لربط معالجات الأحداث الخاصة في jQuery مثل `mouseenter` و `mouseleave`، بالإضافة إلى أيّة معالجات أحداث مخصصة تُنشِئها أنت.

لحذف معالجات أحداث قياسية أو خاصة، يمكننا ببساطة تمرير اسم المعالج أو اسم المعالج الخاص الذي نريد حذفه إلى الدالة `off()`، مثلًا `jQuery('a').off('click')`. إن لم تُمرِّرَ أيّة وسائط إلى الدالة `off()` فسيؤدي ذلك إلى حذف جميع معالجات الأحداث المرتبطة بالعنصر المُحدَّد.

سأريك المفاهيم التي شرحناها في الأعلى في المثال الآتي:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <input type="text" value="click me" />
    <br />
    <br />
    <button>remove events</button>
    <div id="log" name="log"></div>
    <script
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery
.min.js"></script>
    <script>
      (function($)
      {
        // ربط الأحداث
        $('input').on('click', function()
        {
          alert('You clicked me!');
        });
        $('input').on('focus', function()
        {
         // استخدام alert والحدث focus معًا سيسبب بحلقة
         // تكرار لا نهائية، لأن حدث التركيز على حقل
         // الإدخال سيُطلَق في كل مرة نغلق فيها مربع التحذير
         // سنعرض الناتج في الصفحة بدلًا من استخدام alert
          $('#log').html('You focused this input!');
        });

        // استخدام الدالة click كاختصار
        $('button').click(function()
        {
          // فك ارتباط الأحداث
          $('input').off('click');
          $('input').off('focus');
```

```
            // أو فك ارتباط جميع الأحداث
            // $('button').off();
        });
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظات

- توفِّر jQuery عدّة اختصارات لاستخدام الدالة `on()` مع جميع أحداث DOM القياسية، أي باستثناء أحداث jQuery الخاصة مثل `mouseenter` و `mouseleave`. لاستخدام هذه الاختصارات عليك وضع اسم الدالة بدلًا من اسم الحدث.

أمثلة: `click()` و `mouseout()` و `focus()`.

- يمكنك أن تربط عددًا لا محدودًا من الأحداث إلى عنصر DOM وحيد باستخدام jQuery.

- توفِّر jQuery دالةً لمعالجة الأحداث باسم `one()` والتي ستربط حدثًا ما إلى عناصر DOM والذي سيُنفَّذ مرةً واحدةً فقط ثم سيُزال. الدالة `one()` ما هي إلا طريقةٌ مختصرةٌ تُستعمَل فيها الدالتان `on()` و `off()`.

- أُضيفَت الدالتان `on()` و `off()` إلى إصدار 1.7 من jQuery، وكنّا نستعمل بدلًا منهما الدالتين `bind()` و `unbind()`. أُهمِلَ استعمال الدالتين `bind()` و `unbind()` بدءًا من الإصدار 3.0، لذا لا يُنصَح باستخدامهما في التطبيقات الحديثة.

3. استدعاء معالجات الأحداث برمجيًا عبر دوال الأحداث المختصرة

الشكل المختصر لربط معالجات الأحداث إلى عنصر DOM (مثل: `click()` و `mouseout()` و `focus()`) يمكن أن يُستخدَم أيضًا لاستدعاء معالجات الأحداث برمجيًا. لفعل ذلك علينا استدعاء الدالة المختصرة دون أن نُمرِّر إليها دالةً. هذا يعني أننا نستطيع -نظريًا- ربط معالج أحداث إلى عنصر DOM ثم استدعاء ذاك المعالج مباشرةً. سأشرح الفكرة السابقة في المثال الآتي مستعملًا الحدث `click()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <!-- الضغط على هذا العنصر سيؤدي إلى إظهار hi -->
    <a>Say Hi</a>
    <!-- الضغط على هذا العنصر سيؤدي إلى إظهار hi -->
    <a>Say Hi</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // ربط معالج لحدث النقر إلى جميع عناصر <a>
        // ومن ثم استدعاء تلك المعالجات مباشرةً
```

```
        $('a').click(function()
        {
          alert('hi');
        }).click();
        // سيظهر تحذير (alert) مرتين عند تحميل الصفحة
        // وسيظهر أيضًا عند الضغط على أحد عنصرَي <a>
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة

من الممكن أيضًا استخدام الدالة `trigger()` لاستدعاء أحداث معيّنة؛ مثلًا:

```
jQuery('a').click(function ()
{ alert('hi') }).trigger('click')
```

سيعمل ما سبق أيضًا مع الأحداث المخصصة.

4. وحّدَت jQuery طريقة التعامل مع الكائن event

وحّدَت jQuery طريقة التعامل مع الكائن event بناءً على معايير W3C. وهذا يعني أنَّه عندما يُمرَّر الكائن event إلى دالة مُعالِجة للأحداث، فلن تضطر للقلق حول كيفية تعامل المتصفحات مع الكائن event (مثلًا: عبر الكائن `window.event` في متصفح IE). يمكنك استخدام الخاصيات والدوال الآتية التابعة للكائن event دون أن تقلق من الاختلافات بين المتصفحات لأنَّ jQuery قد وحّدَت طريقة التعامل مع الكائن event.

- خاصيات الكائن event
  - event.type
  - event.target
  - event.data
  - event.relatedTarget
  - event.currentTarget
  - event.pageX
  - event.pageY
  - event.result
  - event.timeStamp
- الدوال التابعة للكائن event
  - event.preventDefault()
  - event.isDefaultPrevented()
  - event.stopPropagation()
  - event.isPropagationStopped()
  - event.stopImmediatePropagation()
  - event.isImmediatePropagationStopped()

للوصول إلى الكائن event الموحّد في jQuery عليك تمرير وسيط اسمه «event» (أو اختر أيَّ اسمٍ يحلو لك) إلى دالةٍ مجهولةٍ مُمرَّرةٍ إلى دالةٍ للتعامل مع الأحداث في jQuery. ثم يمكنك استخدام المعامل داخل الدالة المجهولة للوصول إلى الكائن event. المثال الآتي يوضِّح هذا المفهوم:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $(window).on('load', function(event)
        {
          alert(event.type);
        }); // الناتج: "load"
      })(jQuery);
    </script>
  </body>
</html>
```

5. فهم مجالات أسماء الأحداث في jQuery

سيصدف وأن نحتاج إلى ربط أكثر من دالةٍ بحدثٍ وحيد لكائنٍ في DOM.

لنأخذ الحدث `resize` على سبيل المثال. يمكننا عبر jQuery أن نُضيف أيَّ عددٍ نشاء من الدوال لمعالجة الحدث `window.resize`؛ لكن ماذا يحدث عندما نحتاج إلى حذف إحدى تلك الدوال ولكن لا نريد حذفها كلها؟

إذا استخدمنا `$(window).off('resize')` فستُحذَف جميع الدوال المرتبطة بالحدث `window.resize`. أما لو وضعنا الحدث في مجال أسماء (namespace). أي على سبيل المثال: `resize.unique`) فيمكننا حذف دالة بعينها:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $(window).on('resize', function()
        {
          alert('I have no namespace');
        });

        $(window).on('resize.unique', function()
        {
          alert('I have a unique namespace');
        });

       // resize.unique سنحذف الدالة المرتبطة بمعالجة الحدث
        $(window).off('resize.unique')
```

```
      })(jQuery);
    </script>
  </body>
</html>
```

أضفنا في المثال السابق دالتين لمعالجة الحدث `resize`. واستخدمنا مجال أسماء للحدث `resize` الثاني ومن ثم حذفناه مباشرةً باستخدام `off()`. فعلتُ ذلك لكي أوضِّح أنَّنا لم نحذف أوّل دالة. يعطينا استخدام مجال الأسماء القدرة على تسمية وحذف دوال معيّنة مُعالِجة للأحداث حين ترتبط أكثر من دالة لتعالج الحدث نفسه على عنصر DOM نفسه.

إضافةً إلى حذف دالة معيّنة مرتبطة بحدث وعنصر مُحدَّد، يمكننا أيضًا استخدام مجالات الأسماء لاستدعاء دالة معيّنة (عبر استعمال `trigger()`) تعالج حدثًا مُحدَّدًا مرتبطًا بعنصرٍ معين. توجد في المثال الآتي دالتان لمعالجة الحدث `click` للعنصر `<a>`، وسنستدعي إحداهما عبر الاستفادة من مجال الأسماء:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a>click</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
```

```
      {
        $('a').on('click', function()
        {
          alert('You clicked me')
        });

        $('a').on('click.unique', function()
        {
          alert('You Trigger click.unique')
        });
        // استدعاء الدالة المرتبطة بمجال
        // أسماء click.unique فقط
        $('a').trigger('click.unique');
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظات

- لا يوجد حد لعمق أو عدد مجالات الأسماء المستخدمة، مثلًا `resize.layout.headerFooterContent`.

- استخدام مجال الأسماء هو طريقةٌ ممتازةٌ لحماية وحذف واستدعاء الدوال المرتبطة بالأحداث والتي تستعملها إضافةٌ (plugin) ما.

- يمكن استخدام مجالات الأسماء مع الأحداث الخاصة كما هو الحال مع الأحداث القياسية، مثلًا `click.unique` أو `myclick.unique`.

6. ما هو تفويض الأحداث

يعتمد تفويض الأحداث (event delegation) على نشر الأحداث (event propagation). فعندما تضغط على العنصر <a> الموجود داخل العنصر <li> الموجود بدوره داخل العنصر <ul> فإن الحدث click يُطلَق ابتداءً من <a> إلى <li> إلى <ul> وهكذا إلى أن تُطلَق (fires) كلُّ دالةٍ تُعالِج حدثًا لعنصرٍ من «أجداد» (ancestors) العنصر الذي أطلَقَ الحدث.

هذا يعني أنّنا لو أضفنا الحدث click إلى عنصر <ul> ومن ثم ضغطنا على عنصر <a> موجود داخل <ul> فسيؤدي ذلك إلى استدعاء الدالة المُعالِجة للحدث click المرتبطة بالعنصر <ul>. ويمكننا حين استدعاء دالةٍ تُعالِجُ حدثًا ما أن نستخدم الكائن event (تحديدًا الخاصية event.target) لتحديد ما هو العنصر في شجرة DOM الذي أطلق الحدث. أكرِّر أنَّ هذا سيعطينا العنصر الذي سبَّبَ بإطلاق الحدث.

يمكننا بفعل ذلك أن نُضيف دالةً لمعالجة حدثٍ ما لعددٍ كبيرٍ من عناصر DOM عبر تعريف دالةٍ وحيدةٍ. وهذا مفيدٌ جدًا، ويمكننا الاستفادة منه (على سبيل المثال) عندما يكون عندنا جدولٌ فيه 500 سطر، إذ يوجد حدث click مرتبطٌ بكل سطر، وحينئذٍ نستطيع الاستفادة من تفويض الأحداث. انظر إلى الشيفرة الآتية للتوضيح (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <ul>
      <li><a href="#">remove</a></li>
      <li><a href="#">remove</a></li>
```

```
      <li><a href="#">remove</a></li>
      <li><a href="#">remove</a></li>
      <li><a href="#">remove</a></li>
      <li><a href="#">remove</a></li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('ul').click(function(event)
        {
          // ربط دالة لمعالجة الحدث click إلى العنصر <ul>
          // وتمرير الكائن event إليها

          // قيمة event.target هي <a>
          // حذف العنصر <li> باستخدام parent()
          $(event.target).parent().remove();

          // إلغاء السلوك الافتراضي للمتصفح
          // وإيقاف نشر الحدث (propagation)
          return false;
        });
      })(jQuery);
    </script>
```

```
  </body>
</html>
```

حسنًا، ماذا لو ضغطتَ على إحدى النقط التي تظهر قبل عناصر القائمة وليس على الرابط نفسه؟ سيؤدي ذلك إلى حذف العنصر `<ul>`. لماذا؟! لأنَّك عندما تضغط على النقطة فستكون قيمة `event.target` هي `<li>` وليس `<a>`، وبالتالي الدالة `parent()` ستُحدِّد العنصر `<ul>` وتحذفه. يمكننا تحديث الشيفرة السابقة لكي تحذف عناصر `<li>` فقط عندما تضغط على `<a>` بتمرير تعبير لمطابقة العنصر `li` إلى الدالة `parent()`.

```
$(event.target).parent('li').remove();
```

الفكرة المهمة هنا هي أنَّ عليك معرفة ما الذي ضُغِط عليه في العناصر التي تحتوي على عناصر أبناء. ولهذا فعليك أن تتحقق من أنَّ الحدث قد أُطلِقَ من العنصر الذي تتوقعه.

7. تطبيق دوال معالجة الأحداث على عناصر DOM بغض النظر عن تحديث شجرة DOM

يمكننا ربط دوال معالجة الأحداث إلى عناصر DOM الموجودة حاليًا في الصفحة وإلى تلك التي لم نضفها إلى الصفحة بعد باستخدام الدالة `on()`. تَستخدِم الدالةُ `on()` تفويضَ الأحداث (event delegation) للسماح لعناصر DOM المُضافة حديثًا بالاستجابة إلى دوال معالجة الأحداث بغض النظر عن التعديلات أو التغييرات الديناميكية لشجرة DOM. يمكن على سبيل المثال استخدام `on()` لإنشاء زر (button) الذي يُنشِئ زرًا آخر... وهكذا إلى ما لا نهاية:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <button>Add another Button</button>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('body').on('click', 'button', function()
        {
          $(this).after("<button>Add another
Button</button>");
        });
      })(jQuery);
    </script>
  </body>
</html>
```

بعد تفحص الشيفرة السابقة، يجب أن يكون واضحًا لك كيف أنَّ الدالة `on()` قد أدّت إلى تطبيق تفويض الأحداث (event delegation) إلى عنصر أب (العنصر `<body>`) في الشيفرة السابقة) لكي يستطيع أيٌّ عنصر `<button>` موجود في الصفحة الاستجابةَ إلى الحدث `click`.

سنستعمل الدالة `off()` لإزالة التأثير الناتج عن استعمال الدالة `on()`، مثلًا: `$('body').off('click', 'button')`.

الفكرة هنا هي أنَّ الدالة `on()` يمكن أن تُستخدَم لربط الأحداث إلى عناصر DOM التي تُحذَف وتُضاف عبر AJAX، وبهذه الطريقة لن تضطر إلى إعادة ربط الأحداث إلى عناصر جديدة في DOM.

ملاحظة	أُضيفَت الدالة `live()` في إصدار jQuery 1.3 والتي كانت تستعمل لتفويض الأحداث، ولكنها أُهمِلَت (deprecated) في إصدار 1.7 وحُذِفَت في إصدار 1.9. وكان يمكن استخدام الدالة `delegate()` كبديل عنها، حيث تؤدي نفس الغرض تمامًا (وإن اختلفت الصيغة قليلًا)، لكن الدالة `delegate()` قد أُهمِلَت في الإصدار 3.0؛ لذا يجدر بنا استعمال الدالة `on()` والتي أضيفت بدءًا من الإصدار 1.7. خلاصة القول: استعمل الدالة `on()` إذا كنتَ تستعمل إصدارًا أكبر من 1.7، أو استعمل `delegate()` إذا كنتَ تستعمل إصدارًا أقل من 1.7.

8. إضافة دالة لمعالجة أكثر من حدث

من الممكن تمرير أكثر من حدث إلى الدالة `on()` لربطها جميعًا مع دالة وحيدة، وهذا يسمح بكتابة الدالة مرةً واحدةً فقط لمعالجة أكثر من حدث. سنربط -في المثال الآتي- دالةً مجهولةً لمعالجة الأحداث `click` و `keypress`:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
```

```
y.min.js"></script>
    <script>
      (function($)
      {
        // الاستجابة إلى أكثر من حدث
        $(document).on('click keypress', function(event)
        {
          alert('A click or keypress event occured on the document.');
        });
      })(jQuery);
    </script>
  </body>
</html>
```

9. تعطيل السلوك الافتراضي للمتصفح باستخدام preventDefault()

عندما يُضغَط على رابط أو عندما يُرسَل أحد النماذج، فسيستدعي المتصفح الوظيفة الافتراضية المرتبطة بهذه الأحداث. على سبيل المثال، الضغط على رابط <a> سيجعل المتصفح يحاول تحميل قيمة الخاصية href للعنصر <a> الذي تم الضغط عليه في نافذة المتصفح الحالية. لإيقاف المتصفح من إجراء هذه الوظائف الافتراضية نستطيع استخدام الدالة `preventDefault()` التي هي جزءٌ من الكائن الموحّد `event` في jQuery (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a href="http://www.jquery.com">jQuery</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // عدم السماح للمتصفح بالانتقال إلى الصفحة الهدف
        $('a').click(function(event)
        {
          event.preventDefault();
        });
      })(jQuery);
    </script>
  </body>
</html>
```

10. إيقاف نشر الأحداث عبر stopPropagation()

تنتشر الأحداث في شجرة DOM. وعندما يُطلَق حدثٌ ما لأيّ عنصرٍ فسيُطلَق نفس الحدث لجميع العناصر «الأجداد» (ancestor). يخدم هذا السلوك الافتراضي توظيف حلول مثل تفويض الأحداث (event delegation).

يمكننا استخدام الدالة `stopPropagation()` الموجودة في كائن `event` الموحّد في jQuery لمنع هذا السلوك الافتراضي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p><span>stop</span></p>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('p').click(function()
        {
          window.location = 'http://www.jquery.com'
        });
        $('span').click(function(event)
        {
          alert('You clicked a span inside of an p element');
          // احذف السطر الآتي وسيُطلَق الحدث
          // click في العنصر <p> أيضًا!
          event.stopPropagation();
        });
      })(jQuery);
    </script>
```

```
  </body>
</html>
```

لن يؤدي الضغط على العنصر span إلى الانتقال إلى jquery.com وذلك لأننا أوقفنا نشر الأحداث في العنصر <span> (أي أنَّ الحدث click المرتبط بالعنصر <p> لن يُطلَق). جرِّب حذف الدالة stopPropagation() وستجد أنَّ الضغط على العنصر span سيأخذك إلى صفحة jquery.com.

11. إلغاء سلوك المتصفح الافتراضي ونشر الأحداث عبر return false

إعادة القيمة false (أي استخدام التعليمة البرمجية return false) تكافئ استخدام الدالتين preventDefault() و stopPropagation() معًا (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <span><a href="javascript:alert('You clicked me!')"
class="link">click me</a></span>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
```

```
      {
        $('span').click(function()
        {
          // <span> إضافة حدث إلى العنصر
          window.location = 'http://www.jquery.com';
        });
        $('a').click(function()
        {
          // <a> تجاهل النقرات على العنصر
          return false;
        });
      })(jQuery);
    </script>
  </body>
</html>
```

إذا حذفتَ العبارة `return false` في الشيفرة السابقة فستُنفَّذ الدالة `alert()` بسبب سلوك المتصفح الافتراضي حيث ستُنفَّذ العبارة البرمجية الموجودة في الخاصية `href`، وستنتقل أيضًا إلى صفحة jquery.com بسبب إطلاق الحدث `click` في العنصر `span`.

12. إنشاء أحداث خاصة وإطلاقها باستخدام ()trigger

لديك في jQuery إمكانية إنشاء أحداث خاصة بك باستخدام الدالة `on()`، وذلك بتوفير دالة واسم خاص بالحدث إلى الدالة `on()`.

ولأنَّ هذه الأحداث خاصة ولا يعرف المتصفح عنها شيئًا، فالطريقة الوحيدة لاستدعاء تلك

الأحداث تكون عبر إطلاقها برمجيًا باستخدام الدالة `trigger()` التابعة لمكتبة jQuery. تفحّص الشيفرة الآتية لترى كيف عُرِّفَ حدثٌ خاصٌ ثم استدعي عبر الدالة `trigger()`:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>jQuery</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('div').on('myCustomEvent', function() {
         // <div> إضافة حدث خاص إلى العنصر
          window.location = 'http://www.jquery.com';
        });
        $('div').click(function() {
          // الضغط على <div> سيؤدي إلى استدعاء الحدث الخاص
          $(this).trigger('myCustomEvent');
        })
      })(jQuery);
    </script>
  </body>
</html>
```

13. نسخ الأحداث مع عناصر DOM

افتراضيًا لن يؤدي نسخ بُنى DOM باستخدام الدالة `clone()` إلى نسخ الأحداث المرتبطة بها. ولكي تُنسَخ عناصر DOM بالإضافة إلى الأحداث المرتبطة بها، فيجب تمرير القيمة المنطقية `true` إلى الدالة `clone()`. أمعن النظر جيدًا في المثال الآتي:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <button>Add another Button</button>
    <a href="#" class="clone">Add another Link</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('button').click(function()
        {
          var $this = $(this);
          // نسخ العنصر وجميع الأحداث المرتبطة به
          $this.clone(true).insertAfter(this);
          // تغيير النص، وإزالة الحدث
          $this.text('button').off('click');
        });
        $('.clone').click(function()
```

```
        {
          var $this = $(this);
          // نسخ العنصر لكن دون الأحداث المرتبطة به
          $this.clone().insertAfter(this);
          // تغيير النص وإزالة الحدث
          $this.text('link').off('click');
        });
      })(jQuery);
    </script>
  </body>
</html>
```

14. استخدام console لإظهار الأحداث المرتبطة بعناصر DOM

إحدى الخدع المفيدة عند استخدام console هي القدرة على عرض قائمة تفاعلية لجميع الأحداث المرتبطة بعنصر DOM باستخدام jQuery. يرتبط الحدثان `click` و `mouseleave` -في المثال الآتي- بالعنصر `<div>`. وباستخدام console ستتمكن من عرض الأحداث المرتبطة بعنصر `<div>` عبر الوصول إلى معلومات الأحداث (بوساطة `$('div').data('events')`):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>click or hover over this text</div>
    <script
```

```
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('div').on('click mouseleave', function(event)
        {
          alert(event.type);
        });
        // انظر إلى console لرؤية الأحداث
        // المرتبطة بالعنصر <div>
        console.dir($('div').data('events'));
      })(jQuery);
    </script>
  </body>
</html>
```

الخلاصة: تُخزَّن معلومات الأحداث عندما نربطها بأحد العناصر.

15. الحصول على إحداثيات X و Y لمؤشر الفأرة في إطار العرض

بربط الحدث mousemove إلى كامل الصفحة (أي document) فيمكننا الحصول على إحداثيات X و Y لمؤشر الفأرة أثناء تحركه داخل إطار العرض (viewport)، وذلك عبر الحصول على الخاصيتين pageY و pageX للكائن الموحّد event في jQuery (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $(document).mousemove(function(e)
        {
          // الخاصية e.pageX تعطيك إحداثيات X
          // الخاصية e.pageY تعطيك إحداثيات Y
          $('body').html('e.pageX = ' + e.pageX +
            ', e.pageY = ' + e.pageY);
        });
      })(jQuery);
    </script>
  </body>
<html>
```

16. الحصول على إحداثيات X و Y للفأرة نسبةً إلى عنصرٍ آخر

من الضروري أحيانًا أن نحصل على إحداثيات X و Y لمؤشر الفأرة نسبةً إلى عنصرٍ آخر غير إطار العرض (أو كامل المستند). وهذا ما نفعله عادةً عند إنشاء «تلميحات» (tooltips)، حيث يظهر التلميح نسبةً إلى مكان تحريك الفأرة.

يمكن فعل ذلك ببساطة بطرح انزياح العنصر (offset) من إحداثيات X و Y للفأرة نسبةً إلى كامل إطار العرض (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <!-- تحريك الفأرة فوق هذا العنصر سيؤدي إلى إظهار إحداثياتها النسبية -->
    <div
style="margin:200px;height:100px;width:100px;background:#ccc;
padding:20px">
      relative to this
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('div').mousemove(function(e)
        {
          // الحركة تكون نسبةً إلى عنصر div
          // وليس إلى كامل المستند (document)
          var relativeX = e.pageX - this.offsetLeft;
          var relativeY = e.pageY - this.offsetTop;
          $(this).html('releativeX = ' + relativeX +
```

```
            ', releativeY = ' + relativeY);
        });
      })(jQuery);
    </script>
  </body>
</html>
```

الفصل السابع:

jQuery ومتصفح الويب



1. تعطيل القائمة المنسدلة الظاهرة بالضغط على الزر الأيمن للفأرة

يمكن باستخدام JavaScript تعطيل القائمة المنسدلة (contextual menu) التي تظهر عندما نضغط على الزر الأيمن. فعل ذلك سهلٌ جدًا في jQuery. كل ما علينا عمله هو تعطيل الحدث contextmenu:

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $(document).on('contextmenu', function()
        {
          return false;
        });

      })(jQuery);
    </script>
  </body>
</html>
```

2. تمرير نافذة المتصفح

هنالك عددٌ كبيرٌ جدًا من الإضافات لتمرير (scroll) نافذة المتصفح، وفعل ذلك سهلٌ جدًا في jQuery إذا كنا نحتاج إلى التمرير فقط. فبضبط خاصية `scrollTop` في عنصرَي `<html>` و `<body>` فإننا سنتمكن من تحديد مكان التمرير الذي نريد الانتقال إليه أفقيًا أو شاقوليًا.

سأستخدمُ في المثال الآتي الدالة `animate()` للتمرير أفقيًا إلى عنصرٍ معيّن في الصفحة.

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <style>
      li {
        padding-bottom: 500px;
      }
    </style>
    <ul>
      <li><a href="#" class="next">Next</a></li>
      <li><a href="#" class="next">Next</a>/<a href="#"
class="prev">Previous</a></li>
      <li><a href="#" class="next">Next</a>/<a href="#"
class="prev">Previous</a></li>
      <li><a href="#" class="prev">Previous</a></li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
```

```
y.min.js"></script>
    <script>
      (function($)
      {
        $('.next').click(function()
        {
          $('html, body').animate(
            {
              scrollTop:
              $(this).parent().next().find('a').offset().top
            },
            1000);
        });
        $('.prev').click(function()
        {
          $('html, body').animate(
            {
              scrollTop:
              $(this).parent().prev().find('a').offset().top
            },
            1000);
        });
      })(jQuery);
    </script>
  </body>
</html>
```

الفصل الثامن:

الإضافات في jQuery



1. استخدام $ عند إنشاء إضافة

عندما تكتب إضافة jQuery، فيجب عليك تطبيق آلية منع التضارب مع المكتبات الأخرى التي تستعملها في شيفرات jQuery العادية نفسها، وبهذا يجب أن تحتوى جميع الإضافات ضمن مجال خاص (private scope) حيث يمكن أن تستعمل الرمز $ دون الخوف من حدوث تضارب أو نتائج غير متوقعة.

يجب أن تبدو الشيفرة الآتية مألوفةً لديك لأننا استخدمناها في أغلبية الأمثلة في الفصل الأول من هذا الكتاب (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      // لا تستعمل $ هنا لأنّ ذلك ليس عمليًا
      alert(jQuery(document).jquery);
      (function($)
      {
        // يمكننا استخدام $ هنا دون الخوف من التضاربات
        alert($(document).jquery);
      })(jQuery);
    </script>
```

```
  </body>
</html>
```

2. الإضافات الجديدة المُلحَقَة بالكائن jQuery.fn ستصبح جزءًا من دوال jQuery

تُلحَق الإضافات الجديدة بالكائن `jQuery.fn` والذي هو اختصارٌ أو اسمٌ بديلٌ للكائن `jQuery.prototype`. سأريك في المثال أدناه أنَّنا كتبنا إضافة «عدّاد» (counter) وألحقناها بالكائن `jQuery.fn`. وبفعلنا لذلك سنكون قد أنشأنا أوّل إضافة jQuery مُخصّصة والتي يمكن أن تُستعمَل على مجموعة تغليف فيها كائنات DOM.

عمومًا، يَسمَح لنا إلحاق إضافة بالكائن `jQuery.fn` بإنشاء دوال خاصة شبيهة بتلك الموجودة في أساس jQuery. وذلك لأنَّه عندما نُلحِق دالة الإضافة بالكائن `jQuery.fn` فستُضمَّن دالتنا في سلسلة prototype (`$.fn.count = function(){}`) لجميع كائنات jQuery التي تُنشَأ باستخدام دالة `jQuery`. إذا أصبح رأسك يؤلمك مما سبق، فعليك أن تعرف أنَّ إضافة دالة إلى `jQuery.fn` يعني أنَّ الكلمة المحجوزة `this` داخل الدالة التابعة للإضافة ستُشير إلى الكائن `jQuery` نفسه (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="counter1"></div>
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.count = function()
        {
          // تُشير this هنا إلى كائن jQuery
          var $this = this;
          // ضبط العدّاد ليبدأ من الصفر
          $this.text('0');
          // المهلة الزمنية
          var myInterval = window.setInterval(function()
          {
            var currentCount = parseFloat($this.text());
            var newCount = currentCount + 1;
            $this.text(newCount + '');
          }, 1000);
        };
      })(jQuery);

      jQuery('#counter1').count();
    </script>
  </body>
</html>
```

ملاحظة	بإلحاق الإضافة إلى الكائن `jQuery.fn` فإننا نقول أنَّ إضافتنا تود استخدام الدالة `jQuery` لتحديد السياق (context) (أي عناصر DOM). إذا لم تتطلب إضافتك سياقًا مُحدَّدًا (بعبارةٍ أخرى: مجموعة من كائنات DOM) التي يجب أن تُجري عملياتها عليها، فقد لا تحتاج إلى إلحاق هذه الإضافة بالكائن `$.fn`.

3. ستُشير this داخل إضافة إلى كائن jQuery الحالي

عندما تُلحِق إضافةً بالكائن `jQuery.fn` فإن الكلمة المحجوزة `this` داخل دالة الإضافة ستُشير إلى كائن `jQuery` الحالي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="counter1"></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.count = function()
        {
          // jQuery('#counter1') إلى مساوية هنا this
          // jQuery كائن هو الناتج
         alert(this);
```

```
            // الناتج هو عنصر div
            alert(this[0]);
            // الناتج هو " counter1"
            alert(this[0].id);
          };
        })(jQuery);

        jQuery('#counter1').count();
      </script>
    </body>
</html>
```

من المهم أن تفهم ما الذي تُشير إليه الكلمة المحجوزة this داخل دالة الإضافة.

4. تُستخدَم each() للمرور على كائن jQuery وتوفير مرجعية إلى كل عنصر في ذاك الكائن باستخدام this

يمكننا باستخدام each() أن نُنشِئ حلقة تكرار ضمنية لإضافتنا. وهذا يعني أنَّه لو احتوت مجموعة التغليف على أكثر من عنصرٍ واحد، فستُطبَّق دالة الإضافة على كل عنصر في مجموعة التغليف.

يمكننا فعل ذلك باستخدام الدالة each() في jQuery، والتي هي دالةٌ عامةٌ تُستخدَم للمرور على عناصر الكائنات أو المصفوفات. وهي تُسهِّل إنشاء حلقات التكرار. سنمر في المثال الآتي على كائن jQuery نفسه، وستُشير الكلمة المحجوزة this داخل الدالة المُمرَّرة إلى each() إلى كل كائن موجود في مجموعة التغليف (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="counter1"></div>
    <div id="counter2"></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.count = function()
        {
          this.each(function()
          {
            // تُشير this هنا إلى كائن jQuery الحالي
            var $this = $(this);
            // ضبط العدّاد ليبدأ من الصفر
            $this.text('0');
            var myInterval = window.setInterval(function()
            {
              // المهلة الزمنية
              var currentCount = parseFloat($this.text());
              var newCount = currentCount + 1;
              $this.text(newCount + '');
            }, 1000);
```

```
        });
      };
    })(jQuery);

    jQuery('#counter1, #counter2').count();
  </script>
 </body>
</html>
```

استخدام الدالة `each()` ضروريٌ جدًا إن كنت تريد من إضافتك أن تستعمل المرور الضمني (implicit iteration) على عناصر مجموعة التغليف.

5. تُعيد الإضافات عمومًا كائن jQuery لكي نتمكن من متابعة السلسلة باستخدام دوال jQuery الأخرى

من الطبيعي أن تُعيد أغلبيةُ الإضافاتِ كائنَ `jQuery` لكي لا تؤدي الإضافة إلى «كسر» السلسلة. بعبارةٍ أخرى، إذا لم تحتاج في الإضافة إلى إعادة قيمة معيّنة فيجب أن تسمح بإكمال السلسلة لكي تُطبَّق المزيد من الدوال على مجموعة التغليف. سنُعيد في المثال الآتي كائن `jQuery` عبر التعليمة `return this;` لكي لا نكسر السلسلة. لاحظ أنَّني أستخدام الدالتين `parent()` و `append()` بعد أن استدعيث الدالة `count()` الخاصة بالإضافة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
```

```
  <body>
    <div id="counter1"></div>
    <div id="counter2"></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.count = function()
        {
          return this.each(function()
          {
            // each() بعد "this" أو ،jQuery إعادة كائن
            var $this = $(this);
            $this.text('0');
            var myInterval = window.setInterval(function()
            {
              var currentCount = parseFloat($this.text());
              var newCount = currentCount + 1;
              $this.text(newCount + '');
            }, 1000);
          });
        };
      })(jQuery);
      jQuery('#counter1, #counter2')
```

```
        .count()
        // count() تمكّنا من إكمال السلسلة لأن الدالة
        // jQuery أعادت كائن
        .parent()
        .append('<p>Chaining still works!</p>');
    </script>
  </body>
</html>
```

ملاحظة

من الممكن أن تكون دالة الإضافة هادمةً وذلك عند عدم إعادتها لكائن jQuery.

6. خيارات الإضافة الافتراضية

تحتوي الإضافات عادةً على خياراتٍ تُعتبَر أنَّها الضبط الافتراضي الأساسي لآلية عمل الإضافة، وتُستخدَم تلك الخيارات عندما تُستدعى الإضافة. سأُنشِئ في المثال الآتي كائن defaultOptions يحتوي على خاصيةٍ وحيدةٍ (startCount) وقيمةٍ (0). ويُخزَّن هذا الكائن ضمن الدالة count (أي $.fn.count.defaultOptions)، ونفعل ذلك لكي نتمكن من تخصيص تلك الخيارات من خارج الإضافة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
```

```
    <div id="counter1"></div>
    <div id="counter2"></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.count = function()
        {
          return this.each(function()
          {
            var $this = $(this);
             // ضبط قيمة بداية العدّاد إلى 0
            $this.text($.fn.count.defaultOptions.startCount +
              '');
            var myInterval = window.setInterval(function()
            {
              var currentCount = parseFloat($this.text());
              var newCount = currentCount + 1;
              $this.text(newCount + '');
            }, 1000);

          });
        };
```

```
        $.fn.count.defaultOptions = {
          startCount: 100
        };

      })(jQuery);
      jQuery('#counter1, #counter2').count();
    </script>
  </body>
</html>
```

7. خيارات مخصصة للإضافة

يمكن عادةً تجاوز الخيارات الافتراضية للإضافة عبر ضبط خيارات مخصصة. سنُمرِّر في المثال الآتي الكائن `customOption` كمعامل إلى دالة الإضافة. سيُدمَج هذا الكائن مع كائن `defaultOptions` لإنشاء كائن `options` وحيد. سنستخدم الدالة `extend()` التي توفرها jQuery لدمج عدّة كائنات في كائنٍ وحيد. تسمح لنا الدالة `extend()` بإضافة خاصيات (أو قيم) جديدة إلى كائن. ويمكننا الآن تمرير الخيارات إلى دالة الإضافة بعد تغييرنا لشيفرتها. حيث سأمرِّر في المثال الآتي الرقم 500 إلى الدالة `count` لكي يُستخدَم كنقطة بداية للعدّاد، وهذا الخيار المُخصص سيتجاوز القيمة الافتراضية `0` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div id="counter1"></div>
```

```
    <div id="counter2"></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.count = function(customOptions)
        {
          // إنشاء كائن option جديد، وملؤه
          // بخاصيات defaultOptions و customOptions
          var options = $.extend(
          {}, $.fn.count.defaultOptions, customOptions);
          return this.each(function()
          {
            var $this = $(this);
            // ضبط قيمة بداية العداد إلى القيمة الافتراضية
            // أو إلى القيمة المُخصّصة إن مُرِّرَت إلى الإضافة
            $this.text(options.startCount + '');
            var myInterval = window.setInterval(function()
            {
              var currentCount = parseFloat($this.text());
              var newCount = currentCount + 1;
              $this.text(newCount + '');
            }, 1000);
          });
```

```
        };
        $.fn.count.defaultOptions = {
          startCount: 100
        };
      })(jQuery);

      // تمرير خيارات مُخصّصة سيؤدي إلى تجاوز القيم الافتراضية
      jQuery('#counter1, #counter2').count(
      {
        startCount: 500
      });
    </script>
  </body>
</html>
```

8. تجاوز القيم الافتراضية دون تغيير شيفرة الإضافة

لمّا كان من الممكن الوصول إلى الخيارات الافتراضية من خارج الإضافة، فمن الممكن أن تُغيّر قيمتها قبل استدعاء الإضافة. وهذا مفيدٌ إن أردتَ أن تُعرِّف قيمًا للخيارات دون تعديل شيفرة الإضافة. وفعلك لذلك سيُبسِّط عمليات استدعاء الإضافات، لأنَّك تُهيّئ الإضافة لتعمل كما تشاء دون أن تُعدِّل شيفرتها (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
```

```
    <div id="counter1"></div>
    <div id="counter2"></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.count = function(customOptions)
        {
          var options = $.extend(
          {}, $.fn.count.defaultOptions, customOptions);
          return this.each(function()
          {
            var $this = $(this);
            $this.text(options.startCount + '');
            var myInterval = window.setInterval(function()
            {
              var currentCount = parseFloat($this.text());
              var newCount = currentCount + 1;
              $this.text(newCount + '');
            }, 1000);
          });
        };
        $.fn.count.defaultOptions = {
          startCount: 100
```

```
    };
  })(jQuery);

  // تجاوز القيم الافتراضية
  jQuery.fn.count.defaultOptions.startCount = 200;

  // startCount: 0 بدلًا من startCount: 200 سنستخدم
  jQuery('#counter1').count();
  jQuery('#counter2').count(
  {
    startCount: 500
  }); // تجاوز القيم الافتراضية
 </script>
</body>
</html>
```

9. إنشاء العناصر أثناء التنفيذ واستدعاء الإضافات برمجيًا

اعتمادًا على طبيعة إضافتك، قد تهتم بإمكانية استدعاء الإضافة بشكل اعتيـادي (عبر عناصر وأحداث DOM) وبرمجيًا. تخيل أنَّ لدينا إضافة لإنشاء مربع حوار: هنالك أوقاتٌ سيظهر فيها مربع الحوار بناءً على أفعال المستخدم، وفي أحيـانٍ أخرى سنحتاج إلى إظهار مربع الحوار بناءً على أحداثٍ تتعلق بالبيئة أو النظام؛ وفي هذه الحالات يمكنك استدعاء الإضافة دون وجود أيّة عناصر في DOM وذلـك بإنشـاء عنصـرٍ بـرمجيٍ لكي يسـتدعي الإضـافة. سأسـتدعي في الشـيفرة الآتيـة الإضافة `dialog()` عند تحميل الصفحة، وذلك بإنشاء عنصر جديد (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <a href="#" title="Hi">dialog, say Hi</a>
    <a href="#" title="Bye">dialog, say Bye</a>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.fn.dialog = function(options)
        {
          var text = this.attr('title') || this.text();
          alert(text);
        };
      })(jQuery);

      jQuery('a').click(function()
      { // Invoked by user event
        $(this).dialog();
        return false;
      });

      $(window).load(function()
      {
```

```
        // إنشاء عنصر لاستدعاء الإضافة
        jQuery("<a></a>").attr('title', 'I say Hi when
invoked!').dialog(); // الاستدعاء مباشرةً
      });
    </script>
  </body>
</html>
```

كما هو واضح، يمكن أن تكون هنالك عدّة تعديلات على الطريقة السابقة بناءً على خيارات وتعقيد ووظيفة الإضافة. لكن الفكرة الأساسية هنا هي إمكانية استدعاء الإضافات برمجيًا عبر عناصر DOM الموجودة مسبقًا بالإضافة إلى العناصر المُنشأة أثناء التنفيذ.

الفصل التاسع:

تحسين أداء شيفرات jQuery



1. استخدم آخر إصدار من jQuery

فريق تطوير jQuery نشطٌ للغاية ويواصل تحسين شيفرة jQuery من ناحية الأداء. وبالتالي من المهم أن تستعمل دومًا آخر إصدار من jQuery. أنصحك -كمطوِّر jQuery- أن تقرأ سجل التغييرات (release notes) وتضع بالحسبان أنَّ عليك التحديث إلى إصداراتٍ جديدة التي تحتوي على تحسينات كبيرة في الأداء.

2. تمرير «سياق» إلى دالة jQuery سيُحسّن من الأداء

يمكنك أن تُسرِّع قليلًا من تنفيذ طلبية jQuery عندما تُمرِّر سياقًا (context) إلى دالة jQuery، وبفعل لذلك ستُقلِّل عدد عناصر DOM التي ستبحث jQuery فيها؛ وذلك بتمرير وسيطٍ ثانٍ إلى دالة jQuery، الذي هو مرجعية إلى عنصر DOM وحيد، وسيُستخدَم هذا العنصر كنقطة انطلاق لطلبية DOM (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>
      <div>
        <div id="context">
          <a href="#">jQuery</a>
        </div>
      </div>
    </div>
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // "context" :الناتج
        alert($('a',
document.getElementById('context')).context.id);

        // "document" الناتج .لن تحصل على تحسين في الأداء
        alert($('a', '#context').context.nodeName);
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة	لكي تحصل على تحسينٍ فعليٍ للأداء، فعليك أن تُمرِّر مرجعية فعلية لعنصر DOM كوسيطٍ ثانٍ أما تمرير كائن jQuery (مثلًا: `$('#context')`) يتطلب البحث في كامل المستند عن ذاك العنصر، وهذا يفقد هذه التقنية مغزاها.

يمكن تطبيق هذه الطريقة على الحالات التي تتوافر فيها قيمة this؛ وذلك عبر تمرير قيمة this كسياق للبحث فيه بدلًا من البحث في كامل المستند (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>
      <div>
        <div id="context">
          <a href="#">jQuery</a>
        </div>
      </div>
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {

        $('div#context').click(function()
        {
          // الناتج: "object HTMLDivElement"
          alert($('a', this).context);
        }).click();

      })(jQuery);
    </script>
```

```
    </body>
</html>
```

3. فهم كيفية تحسين أداء المُحدِّدات

ليست جميع مُحدِّدات jQuery سواءً عندما نتحدث عن الأداء. فهل جميع المتصفحات ومحرِّكات JavaScript وأنظمة التشغيل وعتاد الحواسيب متساوية؟ إضافةً إلى صعوبة الجزم بأداء المُحدِّدات بشكل قاطع.

ضع ببالك أنَّ أداء المُحدَّدات مرتبطٌ مباشرةً بمجموعةٍ مما يلي:

- نوع المُحدِّد المستخدم.
- تعقيد تعبير التحديد.
- تعقيد وحجم شجرة DOM التي سيتم البحث فيها.
- توفر دالة JavaScript مرتبطة بالمُحدِّد. مثلًا: `getElementsByTagName()` و `getElementById()` و `getElementsByClassName()` و `querySelectorAll()`.
- المتصفح، ونظام التشغيل، وعتاد الحاسوب.

ولوجود هذه المتغيرات، فيجب أن يعتمد أداء المُحدِّدات على قواعد بسيطة وغير مقيّدة:

- حدِّد عناصر DOM باستخدام ID أينما استطعت، فتنفيذ ذلك سريعٌ لدعم جميع المتصفحات للدالة `getElementById()`. أما إذا استخدمتَ اسم فئة CSS، فلا تستعملها

بمفردها؛ وإنما استعمل `$('td.class')` أو `$('#nav.class')`. لكن بعد هذا الحد لن تؤثِّر أنواع المُحدِّدات على الأداء إلا تأثيرًا ضئيلًا.

- تمرير مُحدِّد وحيد إلى دالة jQuery أسرع بكثير من تمرير عدّة مُحدَّدات.
- عمومًا كلما كان تعبير التحديد أبسط كان الأداء أسرع، استخدم مثلًا `$('#fine')` بدلًا من `$('#div div.class .right#fine')`.
- البحث في بُنى DOM البسيطة أسرع بكثير من البُنى الكبيرة والمعقدة. وبالتالي يصبح تمرير «السياق» مهمًا جدًا عندما نتعامل مع بُنى DOM الكبيرة.

بعد معرفتك لهذه المعلومات، لن يصعب عليك الموازنة بين المُحدِّدات التي تُمرِّرها لدالة `jQuery` وبين تحسين أداء الشيفرة. ستصبح المُحدِّدات أسرع بكثير عندما تكون هنالك دالةٌ مكافئةٌ لها موجودةٌ في JavaScript والتي يمكن استعمالها من jQuery (مثلًا: `getElementsByClassName()`).

ملاحظة	في أغلبية الحالات، توفير سياق إلى دالة jQuery سيزيد الأداء كثيرًا مقارنةً بكتابة مُحدَّدات مُحسّنة.

4. تخزين مجموعة العناصر المُحدَّدة التي تُستخدَم أكثر من مرة مؤقتًا

لا تُكرِّر إنشاء طلبيات لنفس العناصر عندما تستطيع أن تملك مرجعيةً إلى نتيجة الطلبية. فلو شئتَ إعادةَ استخدام مجموعةِ تغليفٍ تحتوي عناصر مُحدَّدة مسبقًا، فخزِّن نتيجة المُحدِّد في

متغيرٍ محلي ومن ثم استخدم الكائن المحفوظ في الذاكرة عند الحاجة.

هذا المفهوم مهمٌ جدًا عندما نتعامل مع حلقات التكرار. فيجب أن تُخزِّن مجموعة التغليف في متغيرٍ محلي خارج الحلقة، لكي نتفادى تنفيذ نفس الطلبية في كل تكرار.

```
// مثالٌ بطيء:
// تنفيذ $('ul li') في كل تكرار
for (i = 0; i < $('ul li').length; i++) {
    // تنفيذ $('ul') في كل تكرار
    $('ul').eq(i).text();
}

// مثالٌ سريع:
// تخزين الطلبية
var list = $('ul');
// تخزين طول (عدد عناصر) الطلبية
var listLength = list.find('li').length;
// أخذ قيمة طول الطلبية من الذاكرة
for (i = 0; i < listLength; i++) {
    // أخذ قيمة الطلبية من الذاكرة
    list.eq(i).text();
}
```

5. أبقِ التغييرات المُحدَثة على DOM أقل ما يمكن

من المهم جدًا عند تحسين أداء صفحة الويب أن نُبقي تغييرات شجرة DOM أقل ما يمكن، وعليك دائمًا أن تخبر نفسك عند تعاملك مع DOM أنّه كلما قلّت تعاملاتك معها كلما كان ذلك أفضل.

يظهر هذا المفهوم جليًا عندما يأتي الأمر إلى التعامل مع حلقات التكرار، فأوّل حلقة في الشيفرة الآتية ستُحدِّث DOM بمقدار 100 مرة! وهذا غير مقبول لأنّك لا تحتاج إلا إلى التعامل مع DOM مرةً واحدةً فقط.

```
// مثالٌ بطيء: //
var list = $('ul');
for (i = 0; i < 100; i++) {
    // التعامل مع DOM لمئة مرة //
    list.append('<li>List item ' + i + '</li>');
}

// مثالٌ سريع: //
var listItems = ''; // سلسة نصية فارغة
for (i = 0; i < 100; i++) {
    // سنُنشِئ سلسلة نصية تحتوي على كامل بُنية //
    // DOM التي نريد إضافتها //
    listItems += '<li>List item ' + i + '</li>';
}
```

```
// سنستخدم الدالة html() لتحديث شجرة DOM مرةً واحدةً فقط
$('ul').html(myListItems);
```

استخدمنا في الحلقة الثانية سلسلةً نصية تحتوي على كامل بُنية DOM؛ وبعد ذلك استخدمنا هذه السلسلة النصية لتحديث DOM مرةً واحدةً فقط بتمريرها إلى الدالة `html()`.

ملاحظة	لتسريع تعديل بنى DOM قليلًا، فيمكننا استخدام الخاصية `innerHTML` لعنصر DOM مباشرةً. فبدلًا من `$('div').html(stringOfHTML)` استعمل `$('div')[0].innerHTML = stringOfHTML`. وبهذا ستتخطى التعقيدات الموجودة داخل دالة `html()`.

6. تحسين الأداء عبر تمرير كائن يحتوي على مفاتيح وقيم إلى دوال jQuery

تقبل الدالتان `attr()` و `css()` كائنًا فيه الخاصيات وقيمها، إضافةً إلى زوجٍ وحيدٍ من القيم. سيُحسِّن تمرير كائن من أداء الشيفرة بتقليل عدد كائنات `jQuery` المُنشأة وبتقليل الشيفرة المُكرّرة.

```
// مثالٌ بطيء:
$('a').css('display', 'block');
$('a').css('color', 'red');
$('a').attr('title', 'Title Txt');
$('a').atrr('href', 'http://www.jquery.com');
```

```
// مثالٌ سريع:
$('a')
    .css({'display': 'block', 'color': 'red'})
    .attr({'title': 'Title Txt', 'href':
'http://www.jquery.com'});
```

7. تحسين الأداء بتمرير عدّة مُحدِّدات إلى دالة jQuery

من الممكن تجميع المُحدِّدات مع بعضها وتمريرها إلى دالة jQuery كمعامل وحيد وذلك بفصل تلك المحددات بفاصلة (كما في CSS). سيزيد ذلك من الأداء بتقليل عدد كائنات `jQuery` المُنشأة وبتقليل الشيفرة المُكرّرة.

```
// مثالٌ بطيء:
$('#div1').hide();
$('#div2').hide();
$('#div3').hide();

// مثالٌ سريع:
$('#div1, #div2, #div3').hide();
```

8. تحسين الأداء باستخدام الدوال كسلسلة

لا تجرِ عمليات بحث في DOM دون فائدة. من المنطقي استخدام سلسلة من الدوال على نفس الكائن وسيزيد ذلك من كفاءة الشيفرة. وبفعلنا لذاك سنزيد من الأداء بتقليل تكرار الشيفرات.

```
// مثالٌ بطيءٌ يُنشِئ عدِة نسخ من كائن jQuery
$('div .open').hide();
$('div .close').show();
$('div').fadeIn();

// مثالٌ سريعٌ يستخدم سلسلة دوال
// للتعامل مع مجموعة التغليف الأصلية
$('div')
    .find('.open')
      .hide()
    .end()
      .find('.close')
      .show()
    .end()
      .fadeIn();

// يمكننا بشكلٍ بديل أن نفعل الآتي:
var $div = $(div); // تخزين مجموعة التغليف تخزينًا مؤقتًا
$div.find('.open').hide();
$div.find('.close').show();
$div.fadeIn();
```

9. استخدم حلقة التكرار for عند التعامل مع حلقات التكرار الكبيرة

الدوال وبنى التحكم المُضمَّنة في المتصفح أسرع دومًا! لكن السؤال الذي يجب أن نسأله في أغلبية الحالات هو: بكم أسرع؟ الزيادة في السرعة في أغلبية الحالات تكون ضئيلة. فلهذا

أســتخدمُ الدالــة `each()` في jQuery بــدلًا من اســتخدام حلقــة تكــرار موجــودة في أســاس لغة JavaScript.

القاعدة التي أتبعها هي أنَّ جميع المهام التكرارية الصغيرة (التي تكون أقل من 1000 تكرار) تُناسِب الدالة `each()` تمامًا. واستخدام الدالة `each()` سيُسهِّل من كتابة الشيفرات، وهذا أكثر قيمةً من التحسين الضئيل في الأداء الذي يأتي من استخدام حلقات التكرار المُضمنة في لغة JavaScript في حلقـات التكـرار الصـغيرة. والحـق يُقـال أنـني لا أشـعر بفـارقٍ حقيقي عنـدما أجرِّب الصفحة.

أما عندما يأتي الأمر إلى حلقات التكرار الكبيرة (أكثر من 1000 تكرار) فربما عليك أن تبدأ بقياس الفرق بين تنفيذ حلقات JavaScript وحلقات jQuery. هذا الاختبار مهمٌ جدًا في متصفح Internet Explorer الذي يتعامل ببطء شديد مع جميع حلقات التكرار الكبيرة.

10. تغيير المظهر باستخدام ID و Class بدلًا من تعديل خاصيات style مباشرةً

عندما نتعامل مع خاصيات CSS فمن الأفضل تحديث عنصر DOM وإضافة فئة (class) جديدة أو خاصية `id` بدلًا من تحديث خاصية `style` مباشرةً عبر الدالة `css()` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <style>
```

```
      .newStyles {
        background-color: red;
        display: block;
        height: 100px;
        width: 100px;
      }
    </style>
  </head>
  <body>
    <div></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // مثالٌ بطيء:
        $('div').css(
        {
          'display': 'block',
          'width': '100px',
          'height': '100px',
          'background-color': '#f00'
        });
```

```
         // مثالٌ سريع:
         $('div').addClass('newStyles');
       })(jQuery);
     </script>
   </body>
</html>
```

الفصل العاشر:

المؤثرات في jQuery



1. تعطيل جميع دوال المؤثرات في jQuery

من الممكن تعطيل جميع دوال الحركات والمؤثرات التي توفرها jQuery بإسناد القيمة true إلى الخاصية off كما يلي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div style="height:100px; width:100px; background-
color:red; position:absolute;
left:20px;">
      Try to animate me!
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        jQuery.fx.off = true;
        // سيختفي العنصر مباشرةً دون مؤثرات
        $('div').slideUp();
      })(jQuery);
    </script>
  </body>
</html>
```

عند ضبط الخاصية off إلى true فلن تظهر أيّة حركات في دوال التأثيرات وستظهر أو تختفي مباشرةً باستخدام خاصيات CSS التي هي display:none و display:block. يمكنك إعادة تفعيل الحركات بإسناد القيمة false إلى الخاصية off (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div style="height:100px; width:100px; background-
color:red; position:absolute;
left:20px;">
      Try to animate me!
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        jQuery.fx.off = true;
        // سيختفي العنصر مباشرةً دون حركات
        $('div').slideUp();

        jQuery.fx.off = false; // تفعيل الحركات مرةً أخرى
        $('div').slideDown(); // سيظهر العنصر مع حركات
```

```
      })(jQuery);
    </script>
  </body>
</html>
```

2. فهم آلية عمل دالة الحركات stop()

من الضروري عادةً إيقاف حركة تجري حاليًا قبل البدء بأخرى. على سبيل المثال عند استخدام الأحداث الخاصة mouseenter و mouseleave (أو الدالة hover()) فقد تُفعَّل الحركة على عنصرٍ يتحرك سلفًا نتيجةً لحدث mouseenter أو mouseleave سابق. وبسبب محاولة إجراء مجموعة من الحركات، فسيؤدي ذلك إلى حدوث خطأ واضح في الحركة؛ ولتفادي ذلك استخدم الدالة stop() لإيقاف الحركة الحالية قبل البدء بحركة جديدة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div style="height:100px; width:100px; background-
color:red; position:absolute;
left:20px;">
      Hover over Me!
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
```

```
    <script>
      (function($)
      {
        $('div').hover(
          function()
          {
            $(this).stop().animate({left: 75}, 'fast');
          },
          function()
          {
            $(this).stop().animate({left: 20}, 'fast');
          }
        );
      })(jQuery);
    </script>
  </body>
</html>
```

أزل دوال `stop()` من الشيفرة السابقة وحرِّك الفأرة فوق العنصر عدّة مرات لترى ماذا سيحدث بالحركة. تحريك الفأرة فوق العنصر سيؤدي إلى تراكم الحركات، وهذا ما لا تريده بكل تأكيد.

ملاحظة	بالإضافة إلى إمكانية إيقاف الحركة الحالية للعنصر المُحدَّد، يمكنك أيضًا إيقاف جميع الحركات المتراكمة بتمرير القيمة `true` إلى الدالة `stop()` وهذا سيؤدي إلى إيقاف جميع الأحداث المتراكمة سواءً كانت مفعلةً (أي تجري الآن) أم لا.

3. معرفة إن كان العنصر يخضع إلى حركة عبر animated:

يمكن أن يُستعمَل المُرشِّح الخاص `:animated` لتحديد العناصر التي تخضع حاليًا إلى حركة.

سأستخدم في المثال الآتي هذا المُحدِّد لإضافة نص إلى عنصر `<div>` خاضع للحركة (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div style="height:100px; width:100px; background-
color:red; color:white"></div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        function recursiveAnimate()
        {
          $('div').slideToggle('slow', recursiveAnimate);
```

```
        };

        recursiveAnimate();

        $('div:animated').text('I am animating');

      })(jQuery);
    </script>
  </body>
</html>
```

4. استخدام الدوال show() و hide() و toggle() دون حركة

سيؤدي تمرير معامل إلى الدوال show() و hide() و toggle() إلى إظهار أو إخفاء العناصر مع استخدام حركة عبر تغيير خاصيات CSS التالية: height أو width أو opacity أو margin أو padding. من الممكن أن نوقف الحركات عند إخفاء أو إظهار العناصر، وذلك عند عدم تمرير أيّة وسائط لتلك الدوال، وهذا سيؤدي إلى تغيير كيفية تعديل تلك الدوال لظهور العنصر؛ إذ ستظهر أو ستختفي العناصر المتأثرة مباشرةً دون حركات، وذلك عبر تعديل قيمة خاصية display في CSS (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <style type="text/css">
```

```
      div {
        height: 100px;
        width: 100px;
        background-color: red;
        color: white;
        margin: 5px;
      }
    </style>
  </head>
  <body>
    <div>Click Me (hide animation)</div>
    <div>Click Me (hide no animation)</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // إخفاء مع حركات
        $('div:first').click(function()
        {
          $(this).hide(1000)
        });
        // إخفاء دون حركات
        $('div:last').click(function()
        {
```

```
          $(this).hide()
        });
      })(jQuery);
    </script>
  </body>
</html>
```

ملاحظة

دوال jQuery ‏hide()‎ و show()‎ و toggle()‎ و slideUp()‎ و slideDown()‎ و slideToggle()‎ عندما تُستخدَم على العناصر التي لها القيمة inline لخاصية display، فستُغيّر مؤقتًا إلى display:block خلال مدة الحركة.

5. فهم الحركات المتزامنة وغير المتزامنة

من المهم أن تفهم الفرق بين الحركات التي تحدث معًا في وقتٍ واحد، وبين الحركات التي تتعاقب خلال فترةٍ من الزمن. افتراضيًا، عندما تُستعمَل دوال المؤثرات كسلسلة فستحدث المؤثرات بالتتالي بعضها تلو بعض (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div style="height:100px; width:100px; background-
color:red; position:absolute;
left:20px; border:1px solid #ff9933">
```

```
      Animate me!
    </div>
    <div style="height:100px; width:100px; background-
color:red; position:absolute;
left:20px; top:100px; border:1px solid #ff9933">
      Animate me!
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // كل مؤثر سيحدث على حدة بعد انتهاء
        // المؤثر الذي قبله
        $('div:first')
          .slideUp('slow').slideDown('slow').hide('slow');

        // كل مؤثر سيحدث على حدة بعد انتهاء
        // المؤثر الذي قبله
        $('div:last').animate(
        {
          width: '200px'
        }, 1000).animate(
        {
          borderLeftWidth: '10px'
```

```
        }, 1000);
      })(jQuery);
    </script>
  </body>
</html>
```

أما عند استخدام الدالة `animate()` فيمكننا أن نضبط حدوث المؤثرات في الوقت نفسه بتمرير جميع خاصيات CSS التي نريد تغييرها إلى دالة `animate()` وحيدة. سيزداد في المثال الآتي عرض العنصر `<div>` وسيُضاف له إطار على طرفه الأيسر في نفس الوقت (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div style="height:100px; width:100px; background-
color:red; position:absolute;
left:20px; border:1px solid #ff9933">
      Animate me!
    </div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
```

```
        // سيحدث كلا المؤثرين في نفس الوقت
        $('div').animate(
        {
          width: '200px',
          borderLeftWidth: '10px'
        }, 1000);

      })(jQuery);
    </script>
  </body>
</html>
```

6. الدالة animate() هي الدالة الأساسية

الدالة `animate()` هي الدالة الأساسية ذات المستوى المنخفض (low level) التي تُستخدَم لإنشاء جميع الحركات المضبوطة مسبقًا (مثلًا: `hide()` أو `slideDown()`) وتوفِّر القدرة على تغيير قيمة الخاصية `style` (خلال فترة زمنية).

هذا ما يجب عليك أن تعرفه عندما تستخدم هذه الدالة:

- الخاصيات الوحيدة المدعومة هي الخاصيات التي تقبل قيمًا رقميةً. بكلامٍ آخر، لا يمكنك إنشاء مؤثرات لقيمة الخاصية `backgroundColor` (على الأقل ليس بدون إضافة). وكذلك الأمر للخاصيات التي تأخذ أكثر من قيمة مثل `backgroundPosition`.
- في إصدار 1.2 jQuery وما بعده، يمكنك إنشاء مؤثرات لخاصيات CSS باستخدام `em` و `%` إن أمكن ذلك.

- يمكن إنشاء مؤثرات نسبية باستخدام «=+» أو «=-» أمام قيمة الخاصية. حيث يمكنك استخدام هذه التقنية لتحريك العنصر (إيجابيًا أو سلبيًا) نسبةً إلى مكانه الحالي.
- في إصدار jQuery 1.3 وما بعده، إذا حددتَ القيمة `0` مدةً للحركة، فستحدث الحركة مباشرةً وستتحول العناصر إلى التنسيق النهائي دون حركة.
- كاختصار، إذا مُرِّرَتَ القيمة `toggle` فستُعكَس حركة التأثير من النهاية إلى البداية.
- جميع خاصيات CSS المضبوطة عبر دالة `animate()` وحيدة ستتحرك في نفس الوقت.

7. فهم آلية عمل دوال الاختفاء في jQuery

يجب أن تعي المفاهيم الثلاثة الآتية عند استخدام دوال `fadeIn()` و `fadeOut()` و `fadeTo()`.

- على عكس بقية دوال المؤثرات، دوال الاختفاء ستُعدِّل شفافية العنصر. وسيُفتَرَض عند استخدام هذه الدوال أنَّ العنصر الذي «سيختفي» يملك طولًا وعرضًا.
- دوال الاختفاء ستؤدي إلى إخفاء العنصر بدءًا من شفافيته الحالية.
- استخدام الدالة `fadeOut()` سيؤدي إلى إخفاء العنصر بدءًا من شفافيته الحالية وعندما يصبح شفافًا بنسبة 100% فستُغيّر قيمة الخاصية `display` في CSS إلى `none`.

جميع النقط السابقة موضّحة في الشيفرة الآتية (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
```

```
  <body>
    <!-- يجب أن تملك العناصر التي ستخفى طولًا وعرضًا -->
    <div style="height:100px; width:100px; background-
color:red;"></div>
    <button>Fade the rest of the way</button>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $('div').fadeTo('slow', 0.50);
        $('button').click(function()
        {
          // إخفاء العنصر من الشفافية الحالية إلى 0
          // ثم إخفاء العنصر تمامًا عبر الخاصية display:none
          $('div').fadeOut('slow');
        });
      })(jQuery);
    </script>
  </body>
</html>
```

الفصل الحادي عشر:

تقنية Ajax



1. الدالة ()ajax هي الدالة الأساسية

الدالة `ajax()` هي الدالة الأساسية ذات المستوى المنخفض التي تستخدم لإنشاء طلبات XMLHttpRequest (أي AJAX). جميع دوال AJAX الأخرى في jQuery -مثل `load()`- تستعمل الدالة `ajax()`. الدالة `ajax()` لها الكثير من الميزات وقابلة للتخصيص لإنشاء طلبات XMLHttpRequest تناسب احتياجاتنا.

الكائن XMLHttpRequest هو جزءٌ من تقنية Ajax، ويمكن باستخدام هذه التقنية تمرير البيانات بين المتصفح والخادم، ولن نحتاج إلى تحديث الصفحة عند فعل ذلك. انتشر استعمال الكائن XMLHttpRequest كثيرًا في الفترة الأخيرة مما أدى إلى بناء تطبيقات ويب مثل Google Maps و Gmail التي تستخدم الكائن XMLHttpRequest لعرض أجزاء أخرى من الخريطة أو رسائل البريد الإلكتروني الجديدة دون الحاجة إلى إعادة تحميل كامل الصفحة.

لسوء الحظ، تختلف الواجهة البرمجية للتعامل مع Ajax بين المتصفحات اختلافًا كبيرًا، وهذا يعني أنَّ على المبرمجين أن يأخذوا بالحسبان جميع الاختلافات بين المتصفحات لضمان عمل تقنية Ajax بشكلٍ سليم؛ لكن لحسن الحظ، تأتي jQuery بدعمٍ لتقنية Ajax موحِّدةً الفروقات بين المتصفحات في واجهة برمجية خاصة به، وتوفِّر الدالة `ajax()` التي ذكرناها سابقًا، وتوفِّر jQuery أيضًا دوالًا أخرى مختصرة لأنواع مُحدَّدة من طلبيات XMLHttpRequest. هذه الدوال ما هي إلا اختصاراتٌ للدالة `ajax()`، وهي:

- `load()`
- `get()`
- `getJSON()`

- getScript()
- post()

الفكرة الأساسية هنا هي أنَّ هذه الاختصارات مفيدةٌ في بعض الأحيان، لكنها جميعًا تستخدم ajax() وراء الستار. وعندما تريد استخدام جميع الميزات والتخصيصات التي توفرها jQuery لتقنية AJAX، فعليك حينها استخدام الدالة ajax().

أشهر طريقتان لإرسال الطلبيات إلى الخادم هما GET و POST، ومن المهم فهم الفرق بينهما لاستعمالهما استعمالًا صحيحًا.

الطريقة GET تستعمل للعمليات «غير الهادمة»، أي العمليات التي تتضمن «الحصول» على البيانات من الخادم، وليس تعديلها؛ فمثلًا: نستخدم طلبية GET عند محاولة الحصول على نتائج من أحد محركات البحث. يجدر بالذكر أنَّ طلبيات GET يمكن تخزينها مؤقتًا (cached) في المتصفحات، مما قد يؤدي إلى سلوكٍ غير متوقع إذا لم تضع ذلك بالحسبان.

أما الطريقة POST فهي تستعمل للعمليات «الهادمة»، أي العمليات التي تؤدي إلى تعديل البيانات الموجودة في الخادم. فمثلًا، إذا حَفِظَ المستخدم منشورًا في مدونته فيجب حينئذٍ استخدام الطريقة POST لنقل البيانات؛ يجدر بالذكر أنَّ طلبيات POST لا تُخزَّن مؤقتًا عادةً من المتصفح.

ملاحظة	افتراضيًا، الدالتان ajax() و load() ستستخدمان الطريقة GET في بروتوكول HTTP.

لمزيدٍ من المعلومات والأمثلة عن استعمال الدالة ajax()، يرجى مراجعة التوثيق الرسمي.

2. تدعم jQuery تقنية JSONP العابرة للنطاقات

تقنية JSONP (اختصار للعبارة JSON with Padding) هي تقنية تسمح لمُرسِل طلبية HTTP -عندما تُعاد صيغة JSON- أن يوفِّر اسمًا للدالة التي ستُستدعى ويُمرِّر كائن JSON إليها كوسيط. هذه التقنية لا تستخدم XHR؛ وإنما تستخدم العنصر `script` لكي نتمكن من استقبال وإرسال البيانات من وإلى أيّ موقع. الهدف من هذه التقنية هو الالتفاف على المحدوديات الناجمة عن «سياسة المصدر الواحد» (same-origin policy) في تقنية XHR.

باستخدام دالة jQuery ذات الاسم `getJSON()` يمكننا تحميل بيانات JSON من نطاقٍ (domain) آخر عندما تُضاف الدالة المُعالِجة لطلبية JSONP إلى رابط URL. كمثال، هذا هو رابط URL الذي سنستخدمه للوصول إلى الواجهة البرمجية لموقع flickr.

```
http://api.flickr.com/services/feeds/
photos_public.gne?
tags=resig&tagmode=all&format=json&jsoncallback=?
```

القيمة ? تُستخدَم كاختصار يخبر jQuery أن تستدعي الدالة التي مُرِّرت كوسيط إلى الدالة `getJSON()`. يمكنك وضع اسم الدالة التي تريدها إن لم تشأ استخدام هذا الاختصار.

سأضيف إلى صفحة الويب في المثال الآتي أحدث الصور التي تملك الوسم «resig» عبر استخدام الواجهة البرمجية لموقع flickr. لاحظ أنني أستخدم الاختصار ? لكي تستدعي jQuery الدالةَ التي مررتُها إلى الدالةِ `getJSON()`. الوسيط الذي سيُمرَّر إلى تلك الدالة هو كائن JSON الذي استقبلناه من الموقع الخارجي (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // استخدام ? سيعني استدعاء الدالة
        // التي مررناها إلى الدالة getJSON()

$.getJSON("http://api.flickr.com/services/feeds/photos_public
.gne?tags=resig&tagmode=all&format=json&jsoncallback=?",
          function(data)
          {
            // الوسيط data هو كائن JSON الذي
            // استقبلناه من Flickr
            $.each(data.items, function(i, item)
            {
              $('<img />').attr("src",
                item.media.m).appendTo('body');
              if (i == 30) return false;
            });
          });
      })(jQuery);
```

```
    </script>
  </body>
</html>
```

ملاحظة	راجع توثيق الواجهة البرمجية (API) للخدمة التي تستقبل البيانات منها لكي تعرف الاستخدام الصحيح لها. على سبيل المثال، موقع Flickr يستعمل `jsoncallback=?` بينما !Yahoo و Digg يستخدمان `callback=?`.

3. منع المتصفح من تخزين طلبيات XHR مؤقتًا

عندما تُنشِئ طلبية XHR، فسيُخزِّن متصفح Internet Explorer الرد (response) مؤقتًا. وإذا كان الرد ثابتًا والذي سيصلح لفترةٍ من الزمن، فمن المستحسن تخزينه مؤقتًا. أما إذا كان المحتوى الذي تطلبه هو محتوى ديناميكي وقد يتغيّر في أيّة ثانية، فعليك حينئذٍ أن تتأكّد أنَّ المتصفح لن يُخزِّن الرد مؤقتًا. أحد الحلول الممكنة هو تمرير قيمة فريدة في عنوان URL للطلبية، وبهذا ستتأكد أنَّ كل طلبية يجريها المتصفح هي طلبية فريدة لها رابط URL مختلف.

```
// إضافة عبارة فريدة في نهاية الرابط
jQuery.ajax(
  url:'some.php?nocache='+(new Date()).getTime(),
  type:'POST'
);
```

أحد الحلول الأخرى - وهو حلٌ عامٌ- هو أن تضبط عدم تخزين الطلبيات مؤقتًا في جميع طلبيات Ajax وهذا باستخدام الخيار cache. سنستخدم هنا الدالة ajaxSetup لضبط تعطيل التخزين المؤقت لجميع طلبيات Ajax:

```
$.ajaxSetup ({
  // القيمة هي true افتراضيًا
  // القيمة false تعني إيقاف التخزين المؤقت
  cache: false
});
```

وإن أردتَ الآن تفعيل التخزين المؤقت لطلبيات XHR معيّنة، فيمكنك ببساطة تغيير الخيار cache باستخدام الدالة ()ajax. هذا مثالٌ أجرينا فيه طلبية XHR باستخدام الدالة ()ajax مع تجاوز الضبط الافتراضي وتخزين الرد مؤقتًا:

```
$.ajaxSetup ({
  cache: false
});

jQuery.ajax(
  cache: true,
  url: 'some.php',
  type:'POST'
);
```

الفصل الثاني عشر:

مواضيع متفرقة



1. تخزين البيانات في عناصر DOM

من الممكن عبر الدالة `data()` تخزين قيم JavaScript (أي السلاسل النصية والأعداد والقيم المنطقية [boolean] والكائنات والمصفوفات...) ضمن عناصر DOM. هذا الحل لتخزين البيانات في عناصر DOM أفضل من تخزينها كقيمة لخاصيات العنصر. فمثلًا، علينا تفادي تخزين البيانات في خاصيات `title` و `alt` و `rel`. سأُخزِّن في المثال الآتي اللون الذي تم اختياره من القائمة في عنصر `<ul>`، والذي يمكن أن نحصل على القيمة المُخزَّنة فيه لاحقًا.

```
<!DOCTYPE html>
<html lang="en">
  <style type="text/css">
    .selected {
      background-color: #ffc;
    }
  </style>
  <body>
    <ul>
      <li><a href="#">red</a></li>
      <li><a href="#">orange</a></li>
      <li><a href="#">blue</a></li>
      <li><a href="#">green</a></li>
    </ul>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
```

```
    <script>
      (function($)
      {
        $('a').click(function()
        {
          $this = $(this);
          $ul = $this.closest('ul');
          // تخزين القيمة النصية الموجودة داخل العنصر
          $ul.data('selectedColor', $this.text());
          $ul.find('a').removeClass('selected')
            .end().end().addClass('selected');
          // إظهار اللون المُختار
          alert($ul.data('selectedColor'));
        })
      })(jQuery);
    </script>
  </body>
</html>
```

صحيحٌ أنَّ الدالة `data()` تُستخدَم عادةً على عنصرٌ مُحدَّد عبر دالة jQuery (مثلًا: `$('div').data()`) لكن من الممكن أيضًا استخدام الدالة `data()` بمفردها.

يوضِّح توثيق jQuery كلا الحالتين:

- http://api.jquery.com/data
- http://api.jquery.com/jQuery.data

عند تخزين البيانات في عناصر DOM، فأفضل حل هو استخدام الدالة `data()` الموجودة في jQuery.

2. إضافة دوال جديدة إلى مجال أسماء jQuery

من المفيد عادةً إعادة استخدام مجال الأسماء (namespace) الخاص بمكتبة jQuery (بدلًا من إنشاء مجال أسماء خاص بك) لتعريف الدوال لكي تتفادى كتابة شيفرات موجودة في المجال العام (global scope) لكي تتجنب التضاربات. إضافةً إلى أنَّه من المستحسن تخزين أيّة دوال متعلقة بمكتبة jQuery والتي لا تتطلب مجموعةً من عناصر DOM في مجال أسماء jQuery. يمكنك فعل ذلك بإضافة خاصية جديدة إلى كائن `jQuery`. سأضيف في المثال الآتي دالةً جديدةً باسم `customAlert()` إلى كائن `jQuery` (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        $.customAlert = function(text)
        {
          // jQuery في كائن customAlert تخزين الدالة
```

```
            alert(text);
          };

          // customAlert استدعاء الدالة
          $.customAlert('Hi');
        })(jQuery);
      </script>
    </body>
  </html>
```

إذا كانت لديك عدّة دوال مرتبطة ببعضها، فعليك استضافتها جميعًا في مجال أسماء مختلف لكي تتجنب حدوث فوضى في مجال أسماء jQuery. سأنُشِئ في المثال الآتي الكائن `myDialog` الذي يُخزَّن بدوره في كائن `jQuery`؛ وسأستضيف عدّة دوالٍ داخل الكائن `myDialog` تتعلق به.

```
$.myDialog = { // myDialog إنشاء الكائن
  show: function(){ // Show شيفرة },
  hide: function(){ // Hide شيفرة },
  position: function(){ // Position شيفرة },
  initiate: function(){ // Initiate شيفرة }
};

$.myDialog.initiate()
```

3. حساب قيمة خاصية من خاصيات أحد العناصر

قبل ضبط قيمة عنصرٍ ما، من الممكن حساب القيمة بتمرير دالة إلى الدالة `attr()` بدلًا من تمرير سلسلة نصية كثاني معامل. وهذا يسمح لك بالمرور على العناصر في مجموعة التغليف وإضافة قيم فريدة إليها (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <div>0</div>
    <div>1</div>
    <div>2</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        // المرور على كل عنصر <div> وإعطاؤه قيمة id فريدة
        $('div').attr('id', function(i)
        {
          return 'div' + i
        });
```

```
        // يمكن أن نكتب أيضًا
        /*
        $('div').each(function(i){
          $(this).attr('id', 'div' + i);
        });
        */
      })(jQuery);
    </script>
  </body>
</html>
```

4. استخدام خاصيات CSS أو مكافآتها في JavaScript

عند الحصول أو ضبط خاصيات CSS في عناصر HTML، فأنت أمام خيارين: إما أن تستعمل النسخة التي فيها شرطات (-) من اسم الخاصية (مثلًا: background-color) وإما أن تستعمل خاصية JavaScript المكافئة لها (مثلًا: backgroundColor). أهم ما في الأمر -وبغض النظر عن النوع الذي اخترته- أنَّه يجب تمرير الخاصية كسلسلة نصية (أي أن تضعها بين علامتَي اقتباس) إلى الدالة ()css. شخصيًا، أفضِّل استخدام أسماء الخاصيات التي فيها شرطات لأنها تماثل ما أكتبه باستخدام CSS (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <p style="background-color:#990033">jQuery</p>
```

```
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
      {
        alert($('p').css('background-color'));
        alert($('p').css('backgroundColor'));
      })(jQuery);
    </script>
  </body>
</html>
```

خاصية CSS	خاصية JavaScript
background	background
background-attachment	backgroundAttachment
background-color	backgroundColor
background-image	backgroundImage
background-position	backgroundPosition
background-repeat	backgroundRepeat
border	border
border-bottom	borderBottom
border-bottom-color	borderBottomColor

borderBottomStyle	border-bottom-style
borderBottomWidth	border-bottom-width
borderColor	border-color
borderLeft	border-left
borderLeftColor	border-left-color
borderLeftStyle	border-left-style
borderLeftWidth	border-left-width
borderRight	border-right
borderRightColor	border-right-color
borderRightStyle	border-right-style
borderRightWidth	border-right-width
borderStyle	border-style
borderTop	border-top
borderTopColor	border-top-color
borderTopStyle	border-top-style
borderTopWidth	border-top-width
borderWidth	border-width
clear	clear
clip	clip
color	color
cursor	cursor
display	display

filter	filter
font	font
fontFamily	font-family
fontSize	font-size
fontVariant	font-variant
fontWeight	font-weight
height	height
left	left
letterSpacing	letter-spacing
lineHeight	line-height
listStyle	list-style
listStyleImage	list-style-image
listStylePosition	list-style-position
listStyleType	list-style-type
margin	margin
marginBottom	margin-bottom
marginLeft	margin-left
marginRight	margin-right
marginTop	margin-top
overflow	overflow
padding	padding
paddingBottom	padding-bottom

paddingLeft	padding-left
paddingRight	padding-right
paddingTop	padding-top
pageBreakAfter	page-break-after
pageBreakBefore	page-break-before
position	position
styleFloat	float
textAlign	text-align
textDecoration	text-decoration
textDecorationBlink	text-decoration: blink
textDecorationLineThrough	text-decoration: line-through
textDecorationNone	text-decoration: none
textDecorationOverline	text-decoration: overline
textDecorationUnderline	text-decoration: underline
textIndent	text-indent
textTransform	text-transform
top	top
verticalAlign	vertical-align
visibility	visibility
width	width
zIndex	z-index

5. الوصول إلى محتوى iframe

عندما تُستخدَم الدالة `contents()` على عنصر `iframe` (مثلًا: `$('#myiFrame').contents()`) يحتوي على مستند مُستضاف على نفس النطاق فستحصل على محتوى المستند الموجود في عنصر `iframe`، مما يعطينا وصولًا إلى شجرة DOM التابعة له. وبالتالي يمكننا استخدام jQuery لتعديل عناصر DOM التابعة لعنصر `iframe` كما لو كانت جزءًا من الصفحة الأصلية.

```
// الحصول على قيمة innerHTML للعنصر <body> داخل عنصر iframe
$('iframe').contents().find('body').html();

// ضبط قيمة innerHTML للعنصر <body> داخل عنصر iframe
$('iframe').contents().find('body').html('<p>Hi</p>');
```

ملاحظة	عليك أن تعرف ما هي المحدوديات عند استخدام `iframe` يُشير إلى مستند موجود على نطاقٍ آخر قبل أن تحاول استخدام التقنية السابقة.

6. التحميل المسبق للصور

من الحكمة أن نُحمِّل الصور قبل أن يحتاج المستخدم إليها (ربما للحدث `:hover`). هذه دالةٌ مخصصةٌ في jQuery للتحميل المسبق للصور.

```
(function($) {
    // preloadImages() تعريف الدالة
    // jQuery إضافة دالة خاصة إلى مجال أسماء
    $.preloadImages = function(arrayOfImages) {
        $(arrayOfImages).each(function() {
            // src إنشاء عنصر img وضبط الخاصية
            $('<img/>')[0].src = this;
        });
    };

    // preloadImages() استخدام الدالة
    $(window).load(function() {
        // window.onload تحميل الصور بعد الحدث
        $.preloadImages(['img1.jpg', 'img2.jpg',
          'img3.jpg']);
    });
})(jQuery);
```

يمكن أن تكتَب الدالة السابقة كإضافة، راجع فصل الإضافات في jQuery إن لم تكن طريقة كتابة إضافات مألوفةً لديك.

```
(function($) {
    $.fn.preloadImages = function() {
        // jQuery تمثل كائن this
        return this.each(function() {
```

```
            // تمثل قيمةً في المصفوفة this
            $('<img/>')[0].src = this;
        });
    };

    $(window).load(function() {
        // تمرير مصفوفة إلى jQuery
        $(['img1.jpg', 'img2.jpg', 'img3.jpg'])
          .preloadImages();
    });
})(jQuery);
```

7. التحميل المسبق للوسائط عبر XHR

من الممكن أن تُخزِّن وسائط الصفحة مؤقتًا عبر استخدام طلبيات XHR، وتُسهِّل jQuery من ذلك كثيرًا. الفكرة هنا هي طلب الوسائط في الخلفية (أي بعيدًا عن أعين المستخدمين) بعد اكتمال تحميل الصفحة.

نستطيع طلب الوسائط (`.js` و `.gif` و `.png` و `.jpeg` و `.css`) باستخدام الحدث `load()` ودالة `ajax()` في jQuery، والتي يمكن أن نُخزِّنها مسبقًا قبل أن يطلبها المستخدم. سأريك في ما يلي مثالًا عن شيفرةٍ يمكن أن تُستعمَل لتخزين الملفات مؤقتًا بعد جلبها من الخادم عبر عمليات تجري في الخلفية بعد إكمال تحميل الصفحة.

```
(function($){
  // window.onload انتظار الحدث
  $(window).on("load", function(){
    // ثم تحميل الوسائط وتخزينها مؤقتًا
    $.ajax({ url:"javascript.js", dataType:"text" });
    $.ajax({ url:"image.gif", dataType:"text" });
    $.ajax({ url:"flash.swf", dataType:"text" });
    $.ajax({ url:"styles.css", dataType:"text" });
  });
})(jQuery);
```

تنويه

أبقِ في ذهنك أنَّ الملفات المطلوبة ستُخزَّن مؤقتًا افتراضيًا، لكن يمكنك تمرير وسيط إلى دالة `ajax()` (الذي هو `{cache: false}`) لكي تطلب عدم تخزين عنوان URL المطلوب مؤقتًا. إذًا من الممكن إجبار المتصفح على عدم تخزين الوسائط التي تطلبها؛ ربما ترى أنَّ ذلك عديم الفائدة لأننا نحاول من الأصل تخزين الملفات مؤقتًا، لكن ذلك جديرٌ بالذكر ويمكن أن تستفيد منه في الحالات التي تريد أن يحصل فيها المستخدم على نسخةٍ جديدةٍ من الملف في كل مرة.

8. إضافة فئة CSS إلى عناصر HTML لكي تظهر تلك العناصر في المتصفحات التي عطّلَت JavaScript

يمكننا أن تعرف إن كان متصفح المستخدم قادرًا على تشغيل شيفرات JavaScript عبر استخدام قيمة معيّنة للخاصية class في عناصر HTML. وذلك بإنشاء فئة تُطبَّق على عناصر صفحة HTML في المتصفحات التي تدعم JavaScript فقط، وسنستفيد كثيرًا من هذه التقنية إذا أردنا إخفاء أجزاء من الصفحة إن كان المتصفح يدعم JavaScript. وإن لم يكن يدعمها (أو كانت معطلةً) فلن تُخفى أيّة عناصر. سنخفي في المثال الآتي جميع عناصر <div> إذا كانت JavaScript مفعّلةً (مثال حي):

```
<!DOCTYPE html>
<html lang="en">
  <style>
    .js div {
      display: none;
    }
  </style>
  <body>
    <div>Hide if JavaScript is enabled!</div>
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquer
y.min.js"></script>
    <script>
      (function($)
```

```
      {
        $('html').addClass('js');
      })(jQuery);
    </script>
  </body>
</html>
```